View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by ASU Digital Repository

A Comparative Analysis of Graph Vs Relational Database

For Instructional Module Development System

by

Abir Lal Saha

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved July 2017 by the
Graduate Supervisory Committee:

Srividya Bansal, Chair
Ajay Bansal
Javier Gonzalez Sanchez

ARIZONA STATE UNIVERSITY

August 2017

ABSTRACT

In today's data-driven world, every datum is connected to a large amount of data. Relational databases have been proving itself a pioneer in the field of data storage and manipulation since 1970s. But more recently they have been challenged by NoSQL graph databases in handling data models which have an inherent graphical representation. Graph databases with the ability to store physical relationships between two nodes and native graph processing technique have been doing exceptionally well in graph data storage and management for applications like recommendation engines, biological modeling, network modeling, social media applications, etc.

Instructional Module Development System (IMODS) is a web-based software system that guides STEM instructors through the complex task of curriculum design, ensures tight alignment between various components of a course (i.e., learning objectives, content, assessments), and provides relevant information about research-based pedagogical and assessment strategies. The data model of IMODS is highly connected and has an inherent graphical representation between all its entities with numerous relationships between them. This thesis focuses on developing an algorithm to determine completeness of course design developed using IMODS. As part of this research objective, the study also analyzes the data model for best fit database to run these algorithms. As part of this thesis, two separate applications abstracting the data model of IMODS have been developed - one with Neo4j (graph database) and another with PostgreSQL (relational database). The research objectives of the thesis are as follows: (i) evaluate the performance of Neo4j and PostgreSQL in handling complex queries that will be fired throughout the life cycle of the course design process; (ii) devise an algorithm to

determine the completeness of a course design developed using IMODS. This thesis presents the process of creating data model for PostgreSQL and converting it into a graph data model to be abstracted by Neo4j, creating SQL and CYPHER scripts for undertaking experiments on both platforms, testing and elaborate analysis of the results and evaluation of the databases in the context of IMODS.

DEDICATION

This thesis work is dedicated to my parents who have tirelessly worked throughout the prime of their lives to make the highest possible education available to me. They have always supported me in every thick and thin of my life. I dedicate this research work to my Dad & Mom without whose unconditional support and constant guidance, it would not have been possible.

ACKNOWLEDGMENTS

I am extremely grateful to my advisor Dr. Srividya Bansal for her valuable guidance throughout the thesis as well as my Master's program as a whole. There have been situations where she has uplifted my morale and helped me breeze through the task of achieving all my objectives and completing this research work. Also, I would like to mention my gratitude for the support shown by Dr. Ajay Bansal and Dr. Javier Gonzalez-Sanchez during my research and for serving on my thesis committee.

TABLE OF CONTENTS

Page

CHAPTER Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

Relational databases have been considered as the primary data store house for enterprise software applications and solutions since 1970. They have been instrumental and successful in storing and organizing data owing to their high performance and strong atomicity, consistency, isolation and durability (ACID) compliance. Initially, relational databases came to the picture for abstracting information from offline forms and putting them together in tabular structures [1]. Having done that exceptionally well and with the increasing demand in IT world, relational databases have been widely accepted and put to use for software product development involving high amount of create, read, update and delete (CRUD) operations such as banking, health care applications. But off late, with the emergence of social media, data has been generating in leaps and bounds in the range of zeta bytes in a day. In relational databases, all tabular structures and constraints need to be identified before storing the data in the database [1]. But in today's world, with increasing social media data, millions of logs, and financial transaction logs, it is extremely difficult to identify the proper structure or schema of the data as most of the data are semi-structured or unstructured [2]. In today's data-driven world, there has been abundance of relationships in data models. Entities are connected to other entities and have numerous relationships between them. Abstracting all these relationships and constantly updating the connections with emergence of newer nodes has been a challenge

for relational databases [3]. Traditionally, developers are trained to store in rows and columns of a relational model. But in real life data does not always exist in tabular row and column structure; it exists as objects and the relationships between those different objects. The influx of these types of complex, real-world data are increasing in volume, velocity and variety day by day. This is the reason due to which data relationships are increasing at a faster rate and are even more important than the data itself.

Instructional Module Development System [4] is a web-based tool that guides STEM instructors through the complex task of curriculum design, ensures tight alignment between various components of a course (i.e., learning objectives, content, assessments), and provides relevant information about research-based pedagogical and assessment strategies. The course components should be tightly connected and aligned as per the outcome based education model [5]. Hence, the tightly connected components of IMODS makes the data model graphical in nature. There is a huge scope in gathering insights from the data relationships present in IMODS data model and improve the performance of the system.

1.2 Research Statement

Instructional Module Development System (IMODS) guides STEM instructors through the complex task of course design. While designing the courses, it is crucial to find out inconsistencies in the relationships between different components of the course design. Complex queries are run throughout the different stages of course design keeping the course components tightly aligned as per PC^3 framework [6]. The primary focus of the thesis is

I. To evaluate the performance of graph databases with highly connected data against traditional relational databases. Experiments will be performed using queries involving different level of joins in a highly connected data model. The performance of graph database(Neo4j) on answering queries for varying depth of relationships in Instructional Module Development System (IMODS) would be compared with a relational database(PostgreSQL) and analyze results.

II. To design an algorithm and implement a visualization technique for dynamically calculating the progress of course design completeness in IMODS application.

1.3 Research Hypothesis

Graph database owing to their graph data model performs better in storing and retrieving highly connected data. Graph database (Neo4j) performs better in answering queries having higher depth of relationships between nodes in comparison to relational database (PostgreSQL) with increasing size of data. Also, in order to calculate the completeness of the course design process, it is better to use graph based data store or

graph database that can give inferences and retrieve results faster to dynamically calculate the completeness percentage.

1.4 Need for Graph Database

Due to the ever-increasing size of datasets having connected data, even with faster processor and high-speed networks, the performance of relational database is going down. The main reason that can be attributed to this fact is the performance lag in relational databases while dealing with queries involving connected data or data relationships. In recommendation applications or fraud detection systems, relational databases fail to perform well while handling deeper relationships. For any relationship intensive applications, relational databases need to perform complex join operations which may even lead to deeper level join operations for answering queries. These join operations degrade the performance of the databases which is also termed as SQL strain [2]. Join operations are computed during query time matching corresponding primary key and foreign key of join tables which is quite expensive and compute intensive as the size and level of joins increase. Also, with the increase in size of overall database, for answering join queries, relational databases scan the entire tables for finding the referential integrity constraints to determine the data relationships. This has serious implications on the performance of the database. With more and more connected data thriving in relationship intensive applications like social media application, recommendation engine, fraud detection or shortest path finding systems, increasing number and level of join operations lead to a phenomenon called join bomb in relational databases. Consequently, storing and modeling of connected data in relational databases

involves a lot of complex operations involving slower performance and eventually cutting down the revenue generation from those applications. This is where role of graph database comes into play. With the inherent design of storing entities as nodes and capturing relationships between them if exists as explicit relationship between the two nodes, graph database has been exceptional in handling queries involving higher degree of connectedness among data as they can directly infer from the pre-materialized relationships using constant time graph traversal operations. As part of the thesis, an instructional module development system (IMODS) that focuses on creating a framework for designing courses has been used as a case study application. This application has higher degree of relationships between its different components and gives enough opportunities for verifying the performance of graph database against relational database.

CHAPTER 2

BACKGROUND

2.1 Instructional Module Development System (IMODS)

As per the GOALS 2000: Educate America Act signed in March 1994, every student after the successful completion of a course or subject, must achieve some goals or outcomes in accordance to the outcome based education model [5].This particular concept is also followed in STEM education where instructors, faculty and trainers collaborate to design a course in such a way that it has specific outcomes for each student to meet at the end of the curriculum. Instructional Module Development System or IMODS is such a tool which guides STEM instructors through the complex task of curriculum design, ensures tight alignment between various components of a course (i.e., learning objectives, content, assessments, and pedagogy), and provides relevant information about research-based pedagogical and assessment strategies. IMODS provide professional development with facilitation embedded in its design [4]. The application has been designed based on the research in instructional design area of STEM discipline. IMODS framework [7] has been built to provide the following objectives:

I. To identify omissions of key components like content, assessments etc. in a course design
II. To identify inconsistencies or non-alignment of relationships between different components like learning objective's learning domain and assessment's learning domain etc. as per PC^3 framework [6].

III. To provide guidance to the user in the design process

IV. To identify related strategies for instruction and assessment

Learning objective forms the backbone of IMODS framework. According to Robert Mager (1984) [8] , the definition of a course or learning objective can be defined by three characteristics- Performance – description of what the learner is expected to do, Conditions- description of the conditions under which the performance will occur, Criteria- Description of the level of expertise the learner is expected to attain. Every learning objective has one tightly connected action word which in turn are also connected to domain category which belongs to a learning domain [9] that helps to clearly define the learning objective. But in education domain, it has been experienced that learning objectives are in most of the times not well-defined. It makes it hard for new instructors who have disciplinary training but not necessarily education training to design a well-defined course. According to Blooms' Taxonomy, learning domains have been classified in to three categories namely cognitive(mental), affective (emotional) and psychomotor(physical) [9]. Each of the learning domains are categorized or sub-divided into other categories as shown in Figure 3. Each category is sub-divided into a set of verbs that describes what the learners should can do. For example: Cognitive Domain is divided into six categories as Remembering, Understanding, Applying, Analyzing, Evaluating & Creating (Figure 3). A second dimension is added to the course design which identifies the type of knowledge to be imparted to learners. It is known as Knowledge Dimension [10]. Knowledge Dimension is again classified into four categories namely Factual, Conceptual, Procedural, and Metacognitive. [10]. Therefore, learning objective can be defined by simply combining the subject (the learner), the verb

from the cognitive process dimension (what learners must know how to do), and the object from the knowledge dimension (the knowledge they need to acquire).

As part of development of this framework, an additional characteristic is added which is Content that essentially means the description of the knowledge, skills, and behavior to be attained [5]. Thus, the underlying framework of Instructional Module Development System is built using PC^3 model [6] as shown in Figure 1. Additional components like Assessment Technique is also incorporated in the framework as shown in Figure 2. Course-Content is linked to the content and condition components of the objective. This component along with content is used to validate the list of course topics. Similarly, Content-Pedagogy is linked to performance and content components of the objective. Instructional approaches or techniques should correspond to the level of learning expected and knowledge skill set to be learned. Content and performance are used to validate pedagogical techniques. Course -Assessment techniques correspond to performance and criteria components. This effectively validates the suitability of assessment strategies whether it determines the performance of the learner is equivalent to the competency level expected.

In IMODS application [7], an instructional module or course belongs to an owner and provides many learning objective which are defined by the owner himself. Learning objective is defined using the PC^3 model [6] where performance, content, criteria, and condition are considered for creating the definition of learning objective as shown in Figure 1. Every learning objective belongs to a certain action word category. Action word category in turn has a specific domain category which belongs to one of the three learning domains-cognitive, affective, and psychomotor [7]. Thus, we can see a lot of

relationships are present between the different components of course design. Each of them are highly interconnected with varying degree of mappings such as one to one, one to many, many to many. The contents that are present in a specific course has different knowledge dimensions. Also, assessment techniques that are being assigned to learning objectives to gauge the competence level of a learner has knowledge dimensions. These dimensions are matched against the applied contents to make sure course design strategy is consistent or not. In order to clearly define the learning objectives, it is imperative that the key components like learning objective and assessments assigned to them are tightly aligned which means that the learning domain of learning objectives must match with the assessment's learning domain. Similarly, content's knowledge dimension must be similar to assessment's knowledge dimension. If these alignments are not properly done, the course design will be inconsistent.

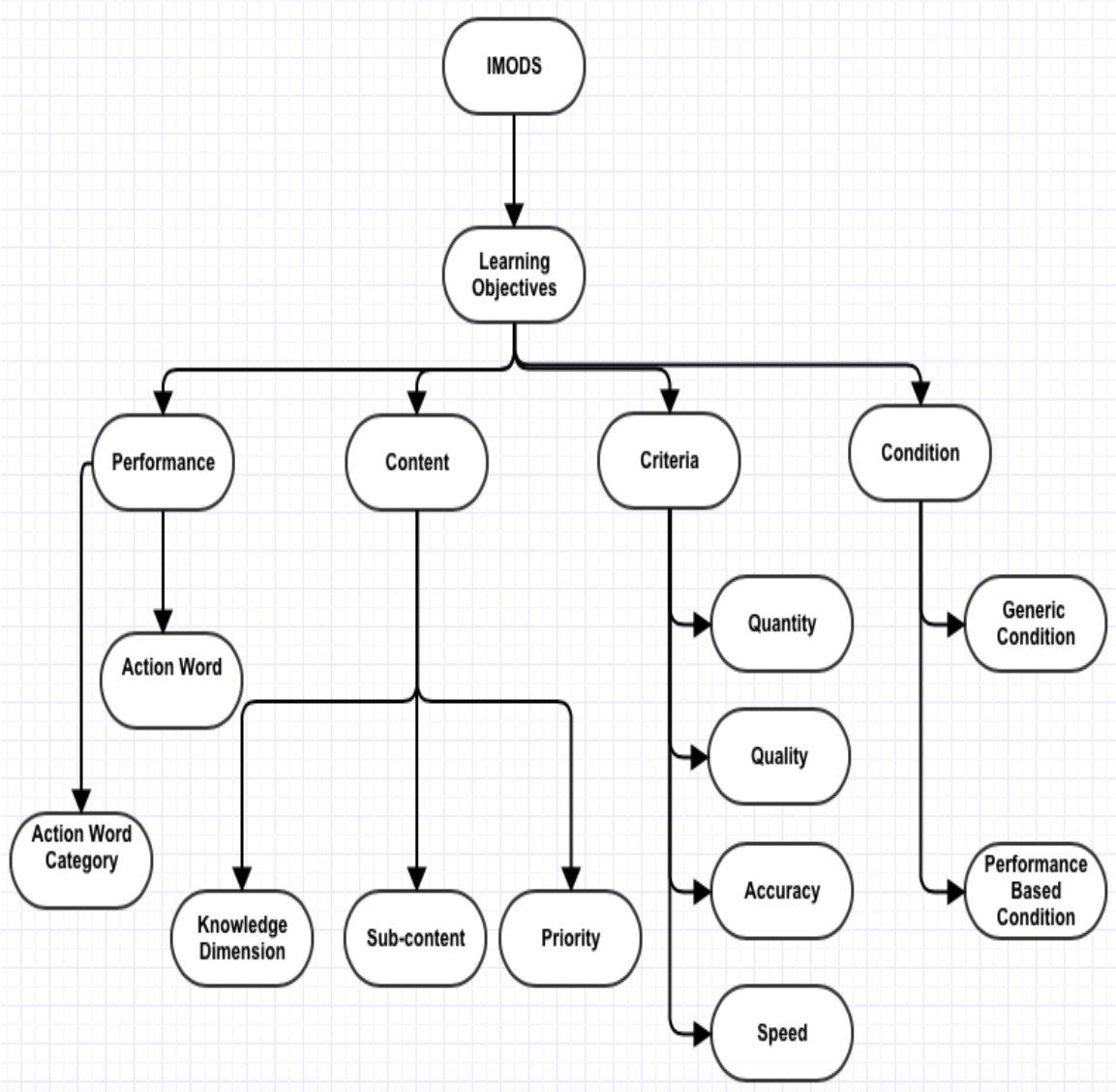

Figure 1: Data Structure Diagram I- IMODS framework

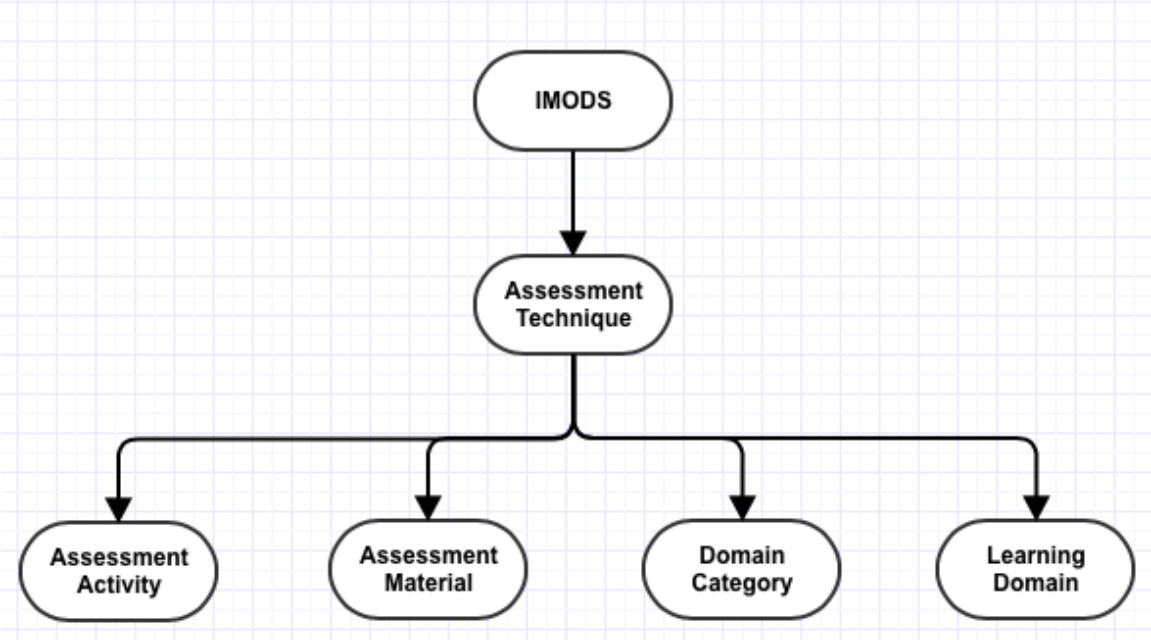

Figure 2: Data Structure Diagram II- IMODS framework

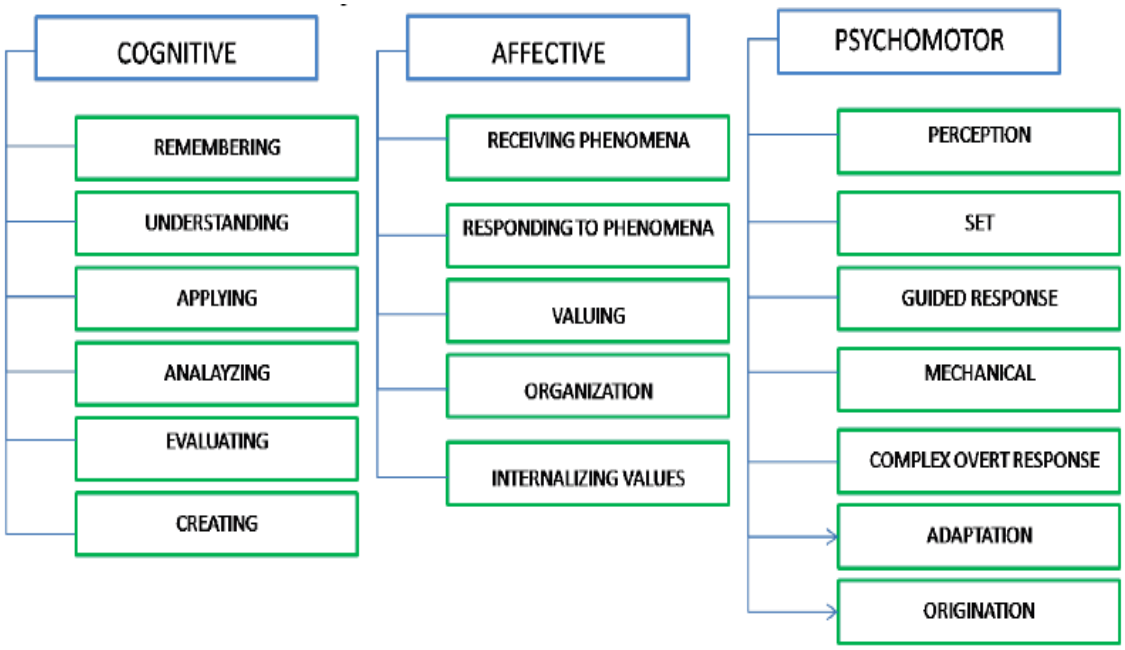

Figure 3: Learning Domains and Domain Categories based on Bloom's Taxonomy [11]
(Blue boxes indicate Learning Domains and Green Boxes indicate Domain Categories)

2.2 Database Paradigm

2.2.1 The Relational Model

Relational Databases came in to existence during 1970 based on the relational model proposed by Edgar Codd [1]. Since then for majority of enterprise applications and software solutions, it has been the predominant paradigm. Initially, it was developed to abstract information from forms and organize them into tabular structure [1]. But gradually it has turned out to be the most popular way of storing, organizing, and managing data because of its well-defined structure and robust data integrity. Data is organized in relational format as attribute and value as shown in Figure 4.

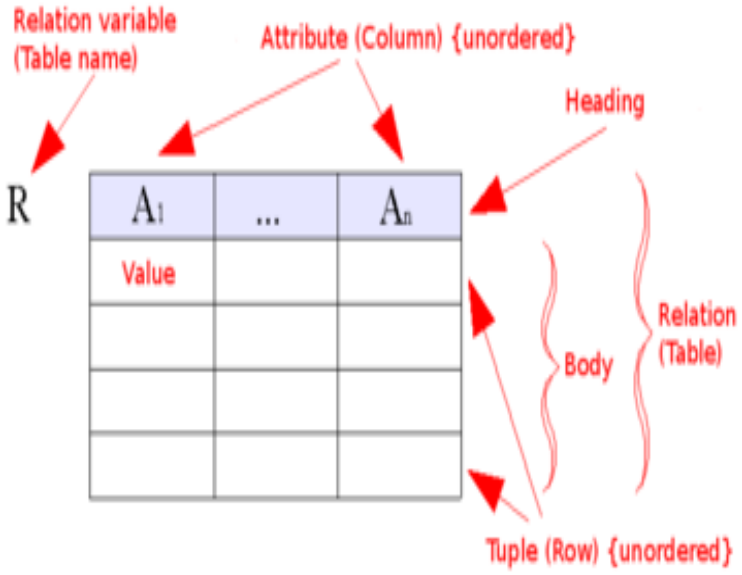

Figure 4: Relational Model

2.2.2 The Graph Model

Graph databases which is a part of NoSQL group of databases is based on the property graph model. The property graph as shown in Figure 5 contains connected entities (nodes) which can hold any number of attributes (key-value-pairs). Nodes can be tagged with labels representing their different roles in your domain. In addition to contextualizing node and relationship properties, labels may also serve to attach metadata—index or constraint information—to certain nodes. Relationships provide directed, named semantically relevant connections between two node-entities. A relationship always has a direction, a type, a start node, and an end node. Like nodes, relationships can have any properties. In most cases, relationships have quantitative properties, such as weights, costs, distances, ratings, time intervals, or strengths [2] .As relationships are stored efficiently, two nodes can share any number or type of relationships without sacrificing performance. In graph model, nodes physical points to all the nodes that it has any connection with. So, in order to perform queries with greater depth of relationships, graph databases need not scan the entire table but can hop from the starting node following the relationships using index free adjacency technique. Instead of using foreign keys to represent a relationship, graph databases use arcs that directly connect two nodes. Operations on this model can be performed through a graph query language. Queries are performed using graph query languages which works on the principle of pattern matching. In case of transactions, graph databases follow a relatively less strict approach than ACID that is known as BASE – Basically Available, Soft state, Eventual Consistency [12].

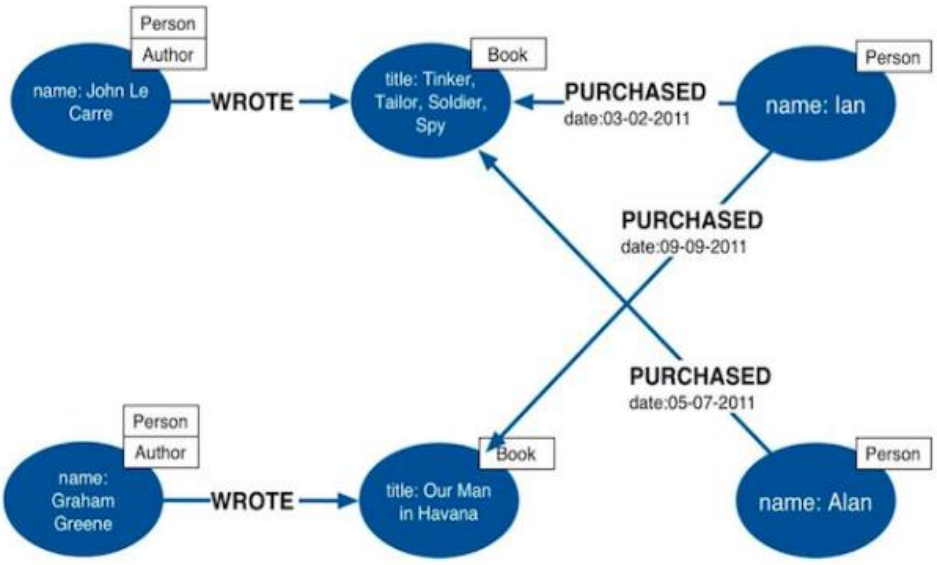

Figure 5: Property Graph Model [2]

2.3 NoSQL Database

With the increasing customer base and data flow in today's data driven world, the need for scalability is the primary concern for every enterprise class software. In order to attain this objective, the phenomenon cloud computing arrived which means provisioning of scalable and elastic IT resources on demand in order to achieve higher scalability, availability and fault –tolerant systems [13]. In cloud aware applications, data is either semi-structured or unstructured owing to the variety, velocity and volume of the data flux. In order to consume this amount of data without any well-defined structure, relational databases were not enough. That is how the birth of NoSQL databases happened by industry stalwarts like Google, Amazon etc. NoSQL databases are available

in many forms such as key-value stores (Redis, Voldemort), document-oriented(MongoDB), columnar store (Cassandra, HBase), graph (Neo4j, Allegro GraphDB).

2.3.1 Graph Databases

Graph Database Management System is a database management system with CRUD operations based on property graph data model. Graph database stores entities as nodes and data relationships are considered as priority citizens. In graph database, there is no need to determine relationships by inferring from foreign key relationships or using map reduce jobs. Since all the nodes physical point to each other, relationships are stored explicitly in the data store. Graph database technology is composed of two important components:

I. Graph Storage – Graph databases uses native graph storage to store nodes and relationships in disks. Some graph database uses underlying relational database to store graph data. Native graph storage performs better while operating on the graph data rather than non-native graph storage with increasing query complexity and data volume [2].

II. Graph Processing – Native graph processing using index-free adjacency is the most efficient way of operating on graph data as nodes physically point to other nodes with an underlying relationship [2].

CHAPTER 3

RELATED WORK

Throughout last decade, lot of work has been done in regard to successfully ascertain the effectiveness of graph databases in software industry. With the emergence of Linked Open Data (LOD) [14], companies and institutions felt the need to share information on the web using Resource Description Format (RDF). According to Linked Data Community, this information previously stored in relational format needed to be converted into graphical model. Several contributions [15], [16], [17] have been made where specifications are given as to how to convert columns(attributes) and their values into key-value pairs as RDF attribute and literals respectively. They focused on mapping the source schema into an ontology using naive transformation technique in which every relational attribute becomes an RDF predicate and every relational value becomes an RDF literal. It was imperative that comparisons should be done between NoSQL graph databases and relational database to understand the need and their efficiency. Some studies have been conducted to compare graph database and relational database from data provenance perspective [18] as well. In this research, Neo4j has been compared with relational databases like MySQL based on parameters like system maturity, ease of coding and security features. The results were varied for various data types as performance of Neo4j came out well for string data but not for integer data [18]. The author concluded that from data provenance perspective it is premature to use graph database in production environment where queries will be on parameters stored in a semi-structured way and less security features in Neo4j contributes to the rejection of graph

database. The version of Neo4j v1.0 database used for comparison was not as mature as MySQL 5.1.42 and the test data used was not realistic [18]. In 2012, another comparison [19] with comparatively smaller dataset of 100 data objects concluded with a result of Neo4j being 2 to 5 times lower in performance than MySQL. The performance of a database also depends on how the query is written and the query language used. In this regard, research [13]has been done to compare the different graph query language like Cypher, Gremlin, and also native access in Neo4j. The performance of Cypher is found out to be slower than Gremlin in FOAF queries and other recommendation queries. In comparison to native object access, Cypher does not perform that well and is about two times slower. But given the readability and easiness of writing queries in Cypher, it proves to be a great choice for using in graph database as query language. Another important aspect is to find the applications or domains where graph databases would be effective. Basically, it has been touted that applications or domains where there is a chance of higher number of relationships or greater depth of relationships, graph databases perform exceedingly well [2]. Work has been done where performances have been measured based on query response time for Neo4j and MySQL databases in the context of cancer treatment application [12]. This research work deals with different categories of datasets and evaluates the performance of different sets of queries under each dataset category [12]. The author concluded that for 1k entries MySQL performance has been far better than Neo4j. But as the dataset increases from 10k to 100k, Neo4j performance improved for queries having greater depth of relationship which would require two or more than two level of JOIN operations in relational database MySQL [12].

There are many e-learning tools like Content Automated Design and Development Integrated Editor (CADDIE) and Intelligent Web Teacher (IWT) available that provides personalized learning services for their users with an ontology based framework using semantic web technologies and RDF data stores [20]. The ontology framework is used to align learning content with teaching strategies. These tools initiates by profiling the learner and suggesting appropriate strategies for providing learning resources to them [20]. They also provide feedback to the instructors for improvements in content and course structure [20]. But they hardly focus on learning objectives or assessment techniques. In IMODS [21] tool, main focus has been given to improve the process of curriculum design by instructors rather than managing the course contents and resources. The existing implementation of IMODS application uses PostgreSQL as database because of better licensing (Open Source- MIT), better performance with sub-queries and better data integrity over other relational databases like MySQL [22]. Given the research objectives, graph database with its explicit storage of relationship feature can prove beneficial and a great fit for IMODS application. But to the best of my knowledge, no research work has been done to evaluate the performance of graph database on a highly connected education tool for course design like IMODS [4]and also no significant amount of work has been done for determining the completeness of course design based on outcome-based education process.

CHAPTER 4

DATA MODELING

4.1. Nature of IMODS Data

In this section, an effort has been made to understand the nature of IMODS data. For this purpose, data from a sample IMOD has been taken and analyzed to understand the structure, complexity and relationships present in the data model. For understanding, sample data as shown in Table 1, 2 and 3 from the software engineering course "Software Enterprise-I: Personal Software Process" in B.S in Software Engineering program at Arizona State University has been taken [5].

4.1.1 Overview of course

Software Enterprise-I: Personal Software Process is a sophomore level course in Software Engineering program. As part of the coursework, students are introduced to object-oriented software design principles using programming languages like Java, software life cycle models, personal software process, process estimation, effort tracking, defect estimation and tracking. A project based pedagogical model is used for delivery of the course [5].

4.1.2 Learning Objective

Software Enterprise-I provides six learning objectives [5] to be accomplished by the students after the completion of the course. The learning objectives are designed

based on the PC^3 framework as shown in Figure 1 and is categorized under Performance, Condition, Criteria, and Content which can be seen in Table 1. The six objectives are enumerated below:

I. LO1: Design a software solution using Object-Oriented Design principles of encapsulation, information hiding, abstraction, inheritance, and polymorphism

II. LO2: Develop a software solution in an object-oriented programming language employing standard naming conventions and making appropriate use of advanced features such as exception handling, I/O operations, and simple GUI

III. LO3: Use object-oriented design tools such as UML class diagrams to model problem solutions and express classes and relationships such as inheritance, association, aggregation, and composition

IV. LO4: Use personal software process for individual development productivity through time estimation and tracking

V. LO5: Use personal software process for individual development quality through defect estimation and tracking

VI. LO6: Demonstrate teamwork

Objective	Learning Domain	Domain Category	Action Category	Action Word Performance	Content	Criteria	Condition
LO1	Cognitive	Create	Plan	Design	Software Solution, Object-oriented Design Principles, Encapsulation, Information Hiding, Abstraction, Inheritance, Polymorphism	Quality	At the completion of this course, student will be able
LO2	Cognitive	Create	Produce	Develop	Software Solution, Object-oriented Programming Language, Standard Naming Convention, Exception Handling, I/O Operations, Simple GUI	Quality, Speed	At the completion of this course, student will be able
LO3	Cognitive	Apply	Implement	Use	Object-oriented Design Tools, UML Class Diagrams, Modelling problem Solutions, Classes, Relationship between classes, Inheritance, Association, Aggregation, Composition	Quality, Speed	At the completion of this course, student will be able
LO4	Cognitive	Apply	Implement	Use	Personal Software Process, Individual Development Productivity, Time	Accuracy (85%)	At the completion of this course, student will be able

					Estimation, Time Tracking		
LO5	Cognitive	Apply	Implement	Use	Personal Software Process, Individual Development, Quality, Defect Estimation, Defect tracking	Accuracy (85%)	At the completion of this course, student will be able
LO6	Cognitive	Apply	Implement	Demonstrate	Teamwork	Quality	At the completion of this course, student will be able

Table 1: Learning Objectives – Software Enterprise-I [5]

4.1.3 Content

In Table 2, all contents for Software Enterprise-I are listed along with sub-contents, priority, and knowledge dimension. This information is crucial for finding out correct assessment and pedagogical techniques that would be beneficial for delivering the content.

Content Topic	Content Sub-Topic	Knowledge Dimension	Priority
Object-Oriented Design Principles	Encapsulation	Factual(F), Conceptual(C)	Critical (3)
	Information Hiding	Factual(F), Conceptual(C)	Critical (3)
	Abstraction	Factual(F), Conceptual(C)	Critical (3)
	Inheritance	Factual(F), Conceptual(C)	Critical (3)
	Polymorphism	Factual(F), Conceptual(C)	Critical (3)
	Software Solution	Conceptual(C), Metacognitive(M)	Critical (3)
Object- Oriented Design Tools	Modelling Problem Solution	Procedural(P), Metacognitive(M)	Critical (3)
	UML Class Diagram	Procedural(P)	Critical (3)
	UML Use Case Diagram	Procedural(P)	Important (2)
Object-Oriented Programming Language	Exception Handling	Factual(F), Conceptual(C)	Important (2)
	I/O Operations	Factual(F), Conceptual(C)	Important (2)
	Simple GUI	Factual(F), Conceptual(C)	Important (2)
	Standard Naming Conventions	Factual(F), Conceptual(C)	Good to know (1)
Personal Software Process	Time Tracking	Factual(F), Conceptual(C)	Critical (3)
	Time Estimation	Factual(F), Conceptual(C)	Critical (3)
	Defect Tracking	Factual(F), Conceptual(C)	Critical (3)
	Defect Estimation	Factual(F), Conceptual(C)	Critical (3)
Teamwork	-	Metacognitive(M)	Important (2)

Table 2: Content Topic– Software Enterprise-I [5]

4.1.4 Assessment Strategies

Both formative and summative assessments [5] have been selected for this course. The underlying PC^3 framework of IMODS aligns all the assignments by ensuring compatibility of learning domains, performance, and criteria. Table 3 enlist all the assessment strategies for this course along with their type, knowledge dimension, criteria etc.

Assessment	Type	Domain Category	Knowledge Dimension	Criteria
Programming Exercise (Write Formal Code)	Formative	Understand, Apply, Analyze, Evaluate, Create	Conceptual, Procedural	Speed, Quality, Accuracy
Partially Guided Programming Exercise	Formative	Understand, Apply, Analyze, Evaluate	Conceptual, Procedural	Quality, Accuracy
Guided Lab Exercise	Formative	Understand, Apply, Analyze, Evaluate	Conceptual, Procedural	Quality, Accuracy
Quiz	Formative	Remember, Understand	Conceptual, Factual	Accuracy, Speed
Project	Summative	Understand, Apply, Analyze, Evaluate, Create	Conceptual, Procedural	Quality, Accuracy
Exam	Summative	Understand, Apply, Analyze, Evaluate	Conceptual, Procedural	Quality, Accuracy

Table 3: Assessment Techniques – Software Enterprise-I [5]

4.2 Analysis of IMODS Relational Database Design

In IMODS, an instructional module can have different learning objectives and for each connection there is a foreign key relationship between imod table and learning_objective table. Each learning objective will have its own action word category which is again connected to domain_category table with foreign key. Domain category belongs to learning domain. Again, for each learning objective, several assessments will be assigned. These assessments can have their own domain category and learning domain as shown in Figure 8. These components should match with learning objective's learning domain in order to justify the alignment of course design components. Similarly, contents will have their own knowledge dimension which must consistent with assessment's knowledge dimension for a correct course design as shown in Figure 9. Instructors can create contents for imods directly and also after creating learning objectives specific contents can be created and added to them as shown in Figure 10. This is the reason circular references between imod, content and learning objective are found in the E-R diagram shown in Figure 10. These types of circular references are intentional in order to give more flexibility to the instructor in adding contents and assessments to the course. Content also show a hierarchical design structure as contents can have sub contents and can be referenced by parent content id. This justifies the presence of a self-loop in Content table as shown in Figure 10. Imod users can create assessments which is stored in imod_user_assessment_technique table without assigning them to learning objectives. These references are again intentional in order to give flexibility to the instructor in designing the course. All these references can be verified from the conceptual diagram of IMODS as shown in Figure 6. The database has been designed conforming to the

requirements of 3NF and have been normalized to an extent. From the E-R diagram of IMODS shown in Figure 7, it can be verified that there is no column in any table that is not dependent on the primary key of that table. All subsets of data that may apply to multiple rows in a table are kept in a separate table. These dependent data have been referenced using foreign keys maintaining referential integrity constraint. For each group of related data, separate tables are created like Imod, Imod_user, Learning objective etc. Indexes have been created on primary keys which are the ids in the tables so that tuples data can be retrieved faster. There was no specific need of output data formatting in IMODS application for security concern and all the attributes of a table can be accessed by the instructor and hence no views have been created.

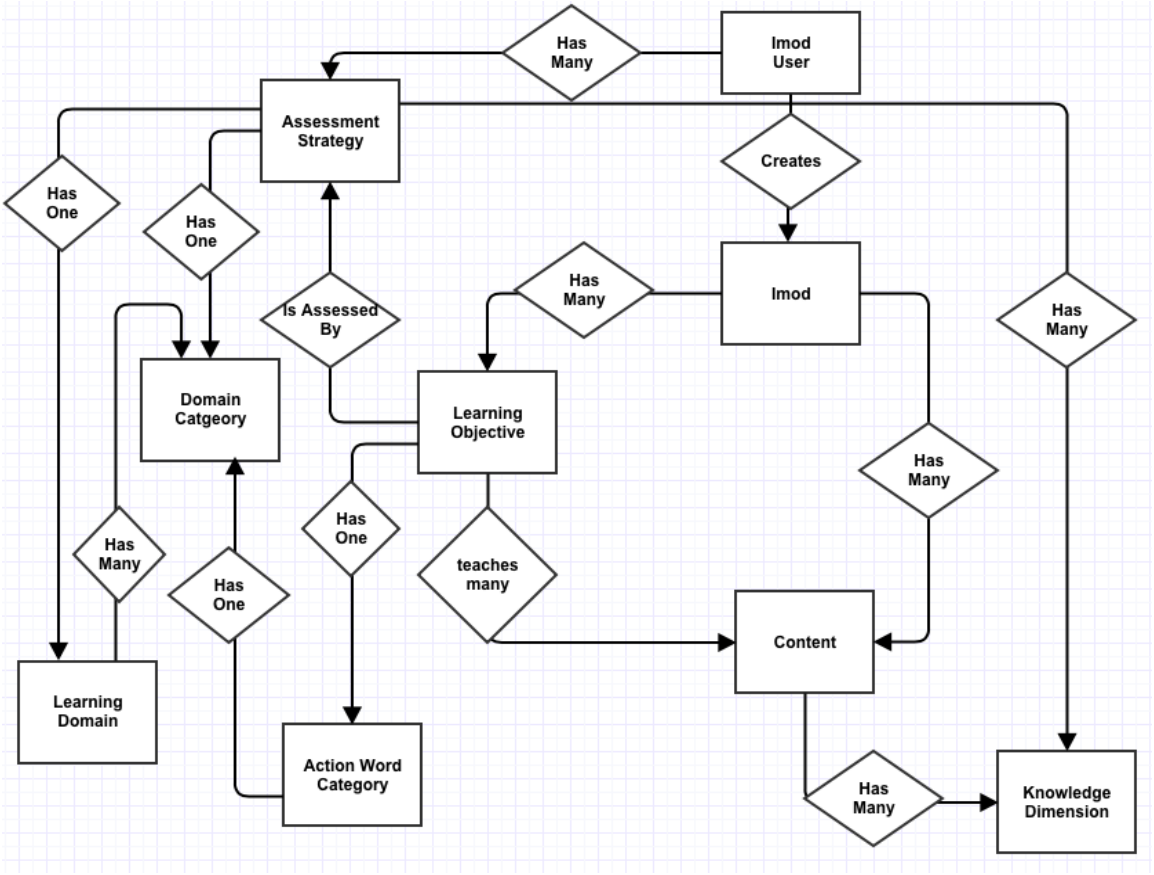

Figure 6: Conceptual Level Diagram of IMODS

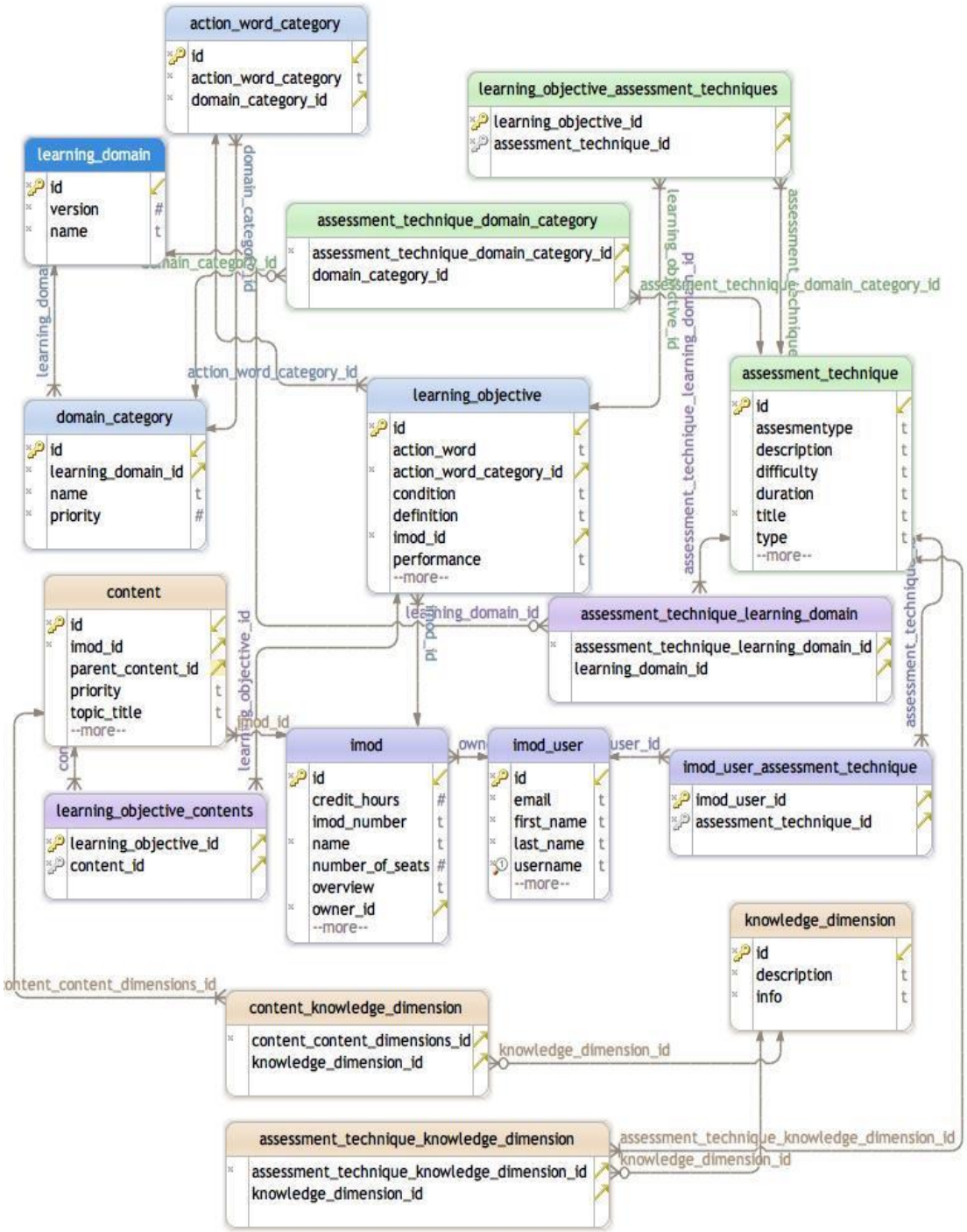

Figure 7: Complete IMODS E-R Diagram

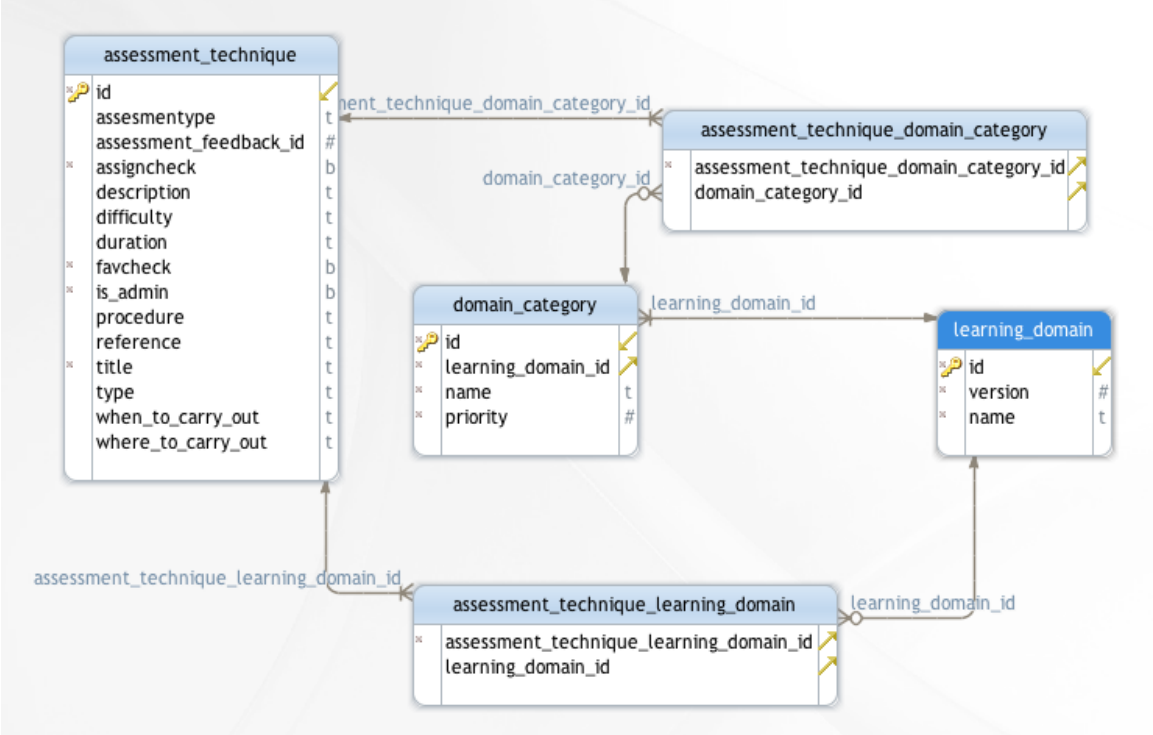

Figure 8: E-R Diagram – Part I

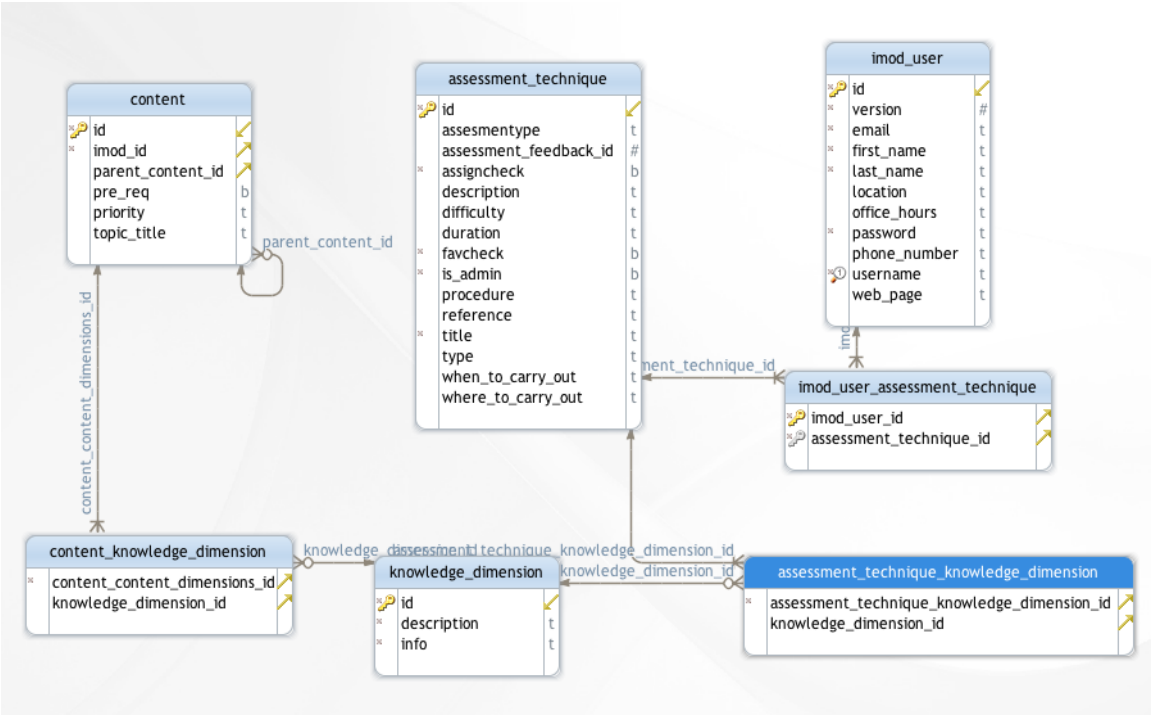

Figure 9: E-R Diagram – Part II

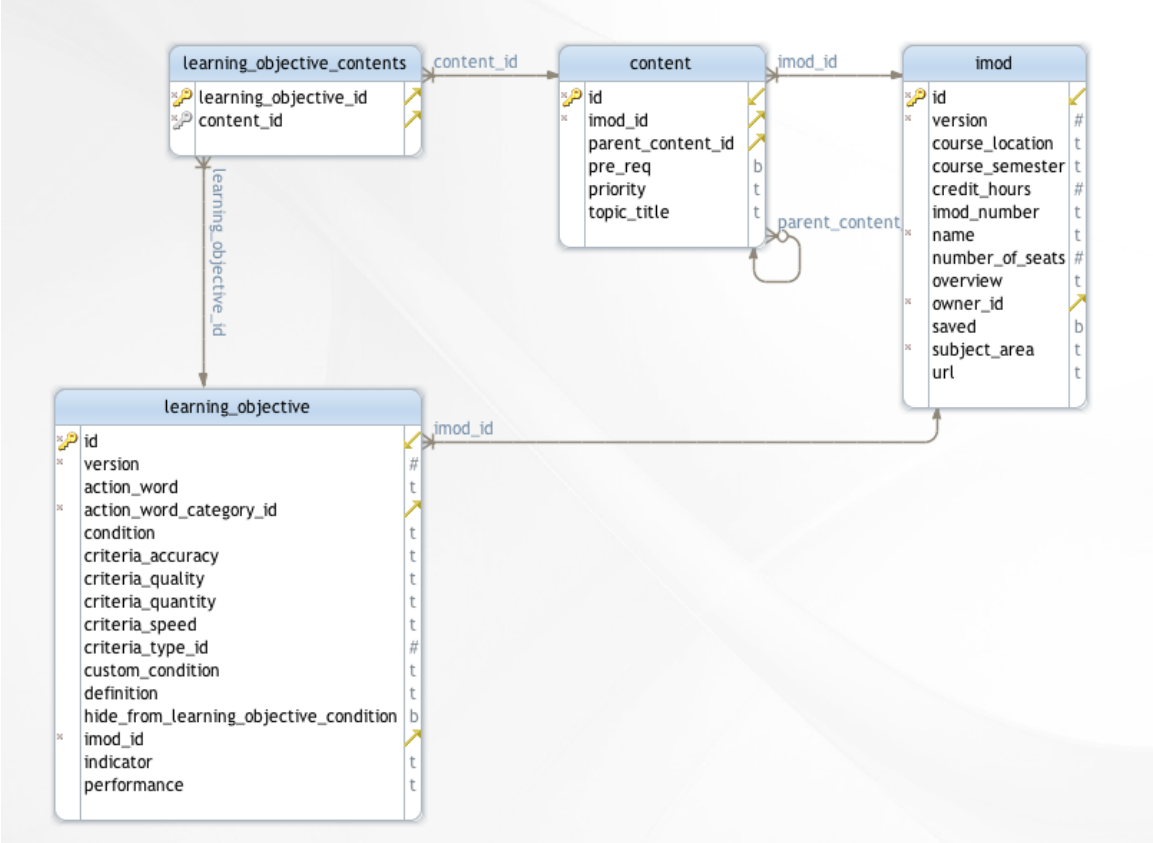

Figure 10: E-R Diagram – Part III

4.3 Graph Data Modelling

On performing data modelling using the Software Enterprise - I course data [5] in alignment with the PC^3 framework, the following graph model as depicted in Figure 11 is generated when implemented in graph database. The key components of the course like learning objectives, content, learning domain, assessment techniques etc. are stored as nodes and related nodes are connected using explicit relationships between them. For this course, six learning objectives are created which are connected to their concerned learning domain, action word category, assessment strategies assigned etc.

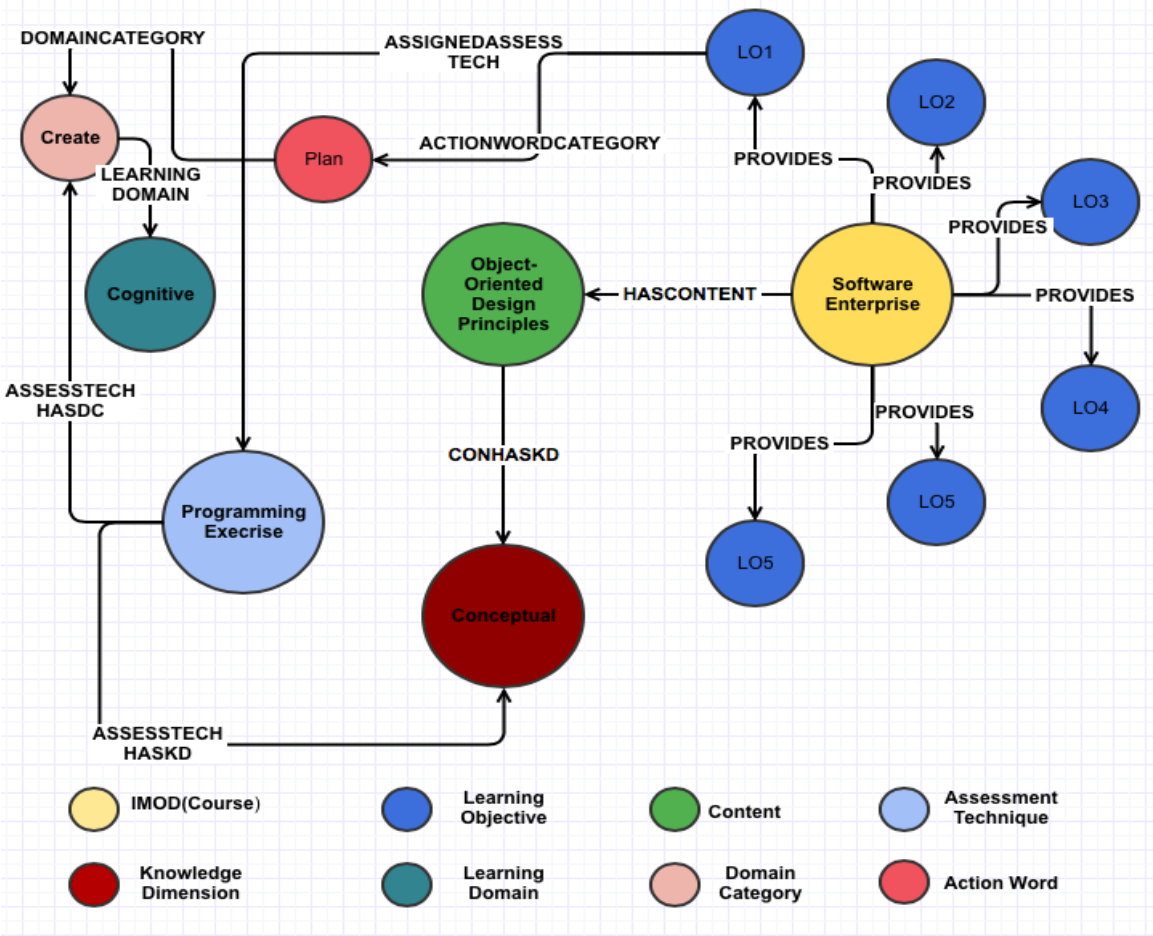

Figure 11: Graphical Data Model for sample course Software Enterprise-I

In the graph model, for the course Software Enterprise-I, the course content Objected Oriented Design has knowledge dimension “Conceptual” which is also connected to the assigned assessment technique “Programming Exercise”. This technique is related to the learning objective “LO1”. Similarly, learning objective has action word “Plan” which belongs to domain category “Create”. The assessment technique belongs to the same domain category “Create” and hence assessment technique’s domain category is consistent with the learning objective’s domain category. From all these relationships shown in Figure 11, we can infer that most of the data are interrelated and connected.

While implementing the IMOD data model in relational database, we need to have several intermediate join tables which helps to create logical relationships between various entities. In a query, which retrieves the relationship between two components, relational databases must find out the logical relationships using the foreign key constraint in the tables. Thus, with increasing volume of dataset, it becomes extremely costly to find out relationships with total table scan. But in graph database, relationships are considered as first-class citizens [2]. Graph database contains entities and explicitly stores relationship between them. If there exists any relationship, then graph database will store it in the disk. It makes it extremely faster to perform graph traversals hopping on to the relationships and moving on to the next connected node. In Table 4, all relationships present in the graphical data model of IMODS is listed.

SL No	Node(P)	Relationship(R)	Node(Q)
1	AssessmentTechnique	ASSESSTECHHASDC	DomainCategory
2	LearningObjective	ACTIONWORD CATEGORY	ActionWordCategory
3	DomainCategory	DOMAINCATEGORY	ActionWordCategory
4	LearningDomain	LDHASDC	DomainCategory
5	DomainCategory	LEARNINGDOMAIN	LearningDomain
6	AssessmentTechnique	KNOWLEDGE DIMENSION	KnowledgeDimension
7	DomainCategory	AWCHASDC	ActionWordCategory
8	AssessmentTechnique	ASSIGNEDASSESS TECH	LearningObjective
9	LearningObjective	CONTENTS	Content
10	DomainCategory	LDHASDC	LearningDomain
11	KnowledgeDimension	KNOWLEDGE DIMENSION	AssessmentTechnique
12	ImodUser	OWNS	Imod
13	Content	HASCONTENT	Imod

14	ActionWordCategory	AWCHASDC	DomainCategory
15	ActionWordCategory	ACTIONWORD CATEGORY	LearningObjective
16	AssessmentTechnique	ASSESSTECHHAS LDOMAIN	LearningDomain
17	Imod	HASCONTENT	Content
18	Imod	IMOD	LearningObjective
19	LearningDomain	LEARNINGDOMAIN	DomainCategory
20	Content	CONTENTS	LearningObjective
21	Imod	OWNS	ImodUser
22	KnowledgeDimension	CONHASKD	Content
23	DomainCategory	ASESSTECHHASDC	AssessmentTechnique
24	LearningObjective	IMOD	Imod
25	ActionWordCategory	DOMAINCATEGORY	DomainCategory
26	Content	CONHASKD	KnowledgeDimension
27	LearningObjective	PROVIDES	Imod

Table 4: Relationships in IMODS graphical data model

CHAPTER 5

DATA GENERATION

5.1 Data Creation & Loading into RDBMS

Test data has been generated using python scripts for testing the performance of the databases. Industry standards and research papers have been read thoroughly before creating the datasets to avoid any sort of bias or inconsistency. Artificial data has been generated imitating the data model of IMODS application including all constraints, relationships, and index. Three different datasets have been generated for 5k,10k and 20k imod users. For a single imod user,0 to 15 imods have been created and assigned randomly. For each imod, 0 to 10 learning objectives and around 0 to 20 contents have been created. Similarly, 0 to 10 assessment strategies have assigned to each learning objectives in a randomized way. The total number of entities and relationships generated for each dataset can be found in tables in section 5.3 Database Statistics below. Python packages like faker and random has been used to generate the artificial data. For tables like action_word_catgory, learning_domain, domain_category, care has been taken so that the generated data represents data according to the Bloom's taxonomy. For every other table, it has been made sure that data is properly distributed and should not be an outlier. Random functions have been used quite often in order to bring uniqueness to the dataset. These datasets have been directly imported in to PostgreSQL Database engine in three different database xDb5k, xDb10k and xDb20k using SQL client pgAdmin. For importing data into graph database Neo4j, Cypher scripts have been used to load the data

in to three different Neo4j database instances YDB5k, YDB10k & YDB20k. After loading the data, cypher scripts are run on the web client of Neo4j for creating specific relationships between different nodes.

5.2 Loading Data into Graph Database

Now for transporting the data from PostgreSQL databases, we have exported the data in to csv format using COPY TO command and for bulk loading the csv files into graphical nodes and relationships in Neo4j, we used LOAD CSV command [2]. This command is a great Extraction-Transform-Load (ETL) tool because of the reasons mentioned below.

I. It supports loading and consuming of data from an URI
II. It directly maps the data into complex graphical/domain structure
III. It has functionality to convert the data types on the fly i.e. data transformation
IV. It supports complex processing and computation
V. It creates and merge data and relationship
VI. It works best for medium to large sized dataset

In conjunction with LOAD CSV, we have used the global query hint USING PERIODIC COMMIT to prevent OutOfMemoryError. Sometimes, while loading large amount of CSV data using LOAD CSV, a single query may fail due to memory

constraint. In such type of situation, we may use PERIODIC COMMIT with a predefined value which gives a hint to the query processor to process only that amount of row in single transaction. Once processed, a new transaction will begin for another transaction for the remaining amount of data. This query hint proves a boon while dealing with 20k users in IMODS application. As we must process relationships in the range of 20000, PERIODIC COMMIT helps to create those relationships without manipulating the configuration of the database engine. Cypher script for creating IMOD user data is shown in Figure 12 and for IMOD data is shown in Figure 13.

Figure 12: Cypher script for loading ImodUser data into Neo4j

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///imod.csv" AS row
MERGE (id:UniqueId{name:'Imod'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:Imod
{__id__:id.count,version:row.version,courseLocation:row.course_location,courseSemester:row.course_semester,creditHours:r
ow.credit_hours,imodNumber:row.imod_number,name:row.name,numberOfSeats:row.number_of_seats,overview:row.overview,owner:r
ow.owner_id,saved:row.saved,subjectArea:row.subject_area,url:row.url})
```

Figure 13: Cypher script for loading Imod data into Neo4j

We know that every imod(course) belongs to an owner or imod user. So, there is a relationship between each imod node with an imod_user node. This relationship is created using the MERGE/ CREATE command of Cypher queries as shown in Figure 14.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///imod.csv" AS row
MATCH (imod:Imod {__id__: toInt(row.id)})
MATCH (user:ImodUser {__id__: toInt(row.owner_id)})
MERGE (user)-[:OWNS]->(imod);
```

Figure 14: Cypher script to create relationship between Imod and ImodUser

After running this script, relationships between all imod and imod users in the dataset have been created. On running a cypher query to return all those relationships, it returns a sub graph showing relationship between Imod and Imod user as shown in Figure 15.

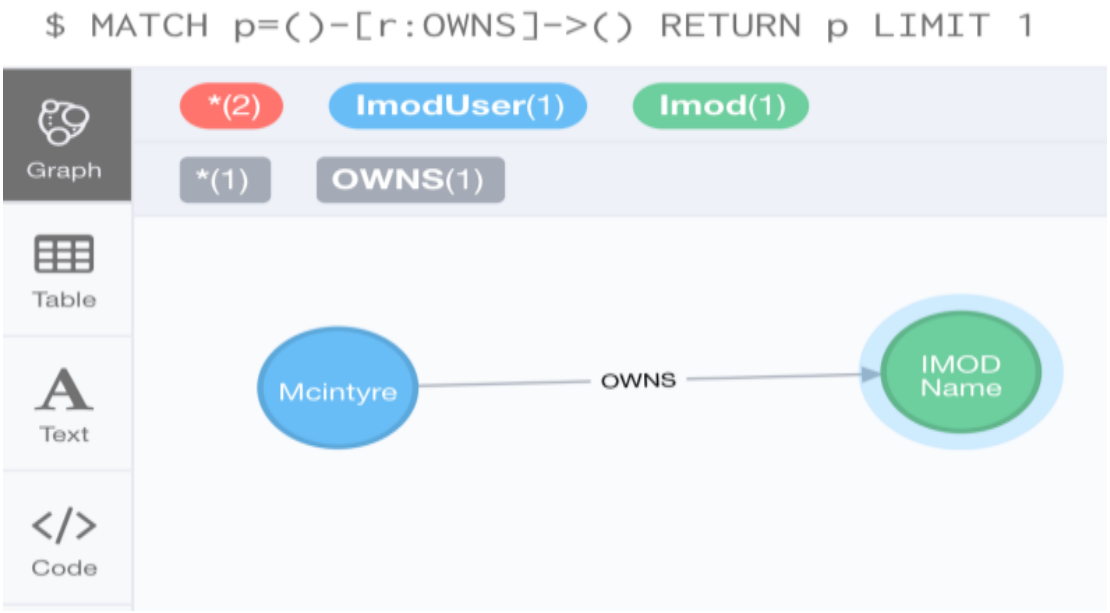

Figure 15: Sub-graph showing relationship between Imod and ImodUser

5.3 Database Statistics

In this section, node and relationship counts are listed for 5k dataset in Table 5 and 6, for 10k dataset in Table 7 and 8 and for 20k dataset in Table 9 and 10 respectively.

SL No.	Dataset	Node Type	Count
1	5k	Imod	5000
2	5k	ImodUser	5000
3	5k	LearningObjective	5028
4	5k	Content	5000
5	5k	AssessmentTechnique	5000
6	5k	ActionWordCategory	59
7	5k	DomainCategory	18
8	5k	KnowledgeDimension	4
9	5k	LearningDomain	3
10	5k	UniqueId	9

Table 5: Neo4j Database YDB5k

SL No	Node(P)	Relationship(R)	Node(Q)	Count
1	AssessmentTechnique	ASSESSTECH HASDC	DomainCategory	5000
2	LearningObjective	ACTIONWORD CATEGORY	ActionWordCategory	5028
3	DomainCategory	DOMAIN CATEGORY	ActionWordCategory	58
4	LearningDomain	LDHASDC	DomainCategory	18
5	DomainCategory	LEARNING DOMAIN	LearningDomain	18
6	AssessmentTechnique	KNOWLEDGE DIMENSION	KnowledgeDimension	5000
7	DomainCategory	AWCHASDC	ActionWordCategory	58
8	AssessmentTechnique	ASSIGNED	LearningObjective	4999

		ASSESTECH		
9	LearningObjective	CONTENTS	Content	4932
10	DomainCategory	LDHASDC	LearningDomain	18
11	KnowledgeDimension	KNOWLEDGE DIMENSION	AssessmentTechnique	5000
12	ImodUser	OWNS	Imod	5000
13	Content	HASCONTENT	Imod	5000
14	ActionWordCategory	AWCHASDC	DomainCategory	58
15	ActionWordCategory	ACTIONWORD CATEGORY	LearningObjective	5028
16	AssessmentTechnique	ASSESSTECH HASLDOMAIN	LearningDomain	5000
17	Imod	HASCONTENT	Content	5000
18	Imod	IMOD	LearningObjective	5028
19	LearningDomain	LEARNING DOMAIN	DomainCategory	18
20	Content	CONTENTS	LearningObjective	4932
21	Imod	OWNS	ImodUser	5000
22	KnowledgeDimension	CONHASKD	Content	5000
23	DomainCategory	ASESSTECH HASDC	AssessmentTechnique	5000
24	LearningObjective	IMOD	Imod	5028
25	ActionWordCategory	DOMAIN CATEGORY	DomainCategory	58
26	Content	CONHASKD	KnowledgeDimension	5000
27	LearningObjective	PROVIDES	Imod	5028

Table 6: Neo4j Database YDB5k Relationships

SL No.	Dataset	Node Type	Count
1	10k	Imod	10000
2	10k	ImodUser	10000
3	10k	LearningObjective	10068
4	10k	Content	10001
5	10k	AssessmentTechnique	10000
6	10k	ActionWordCategory	59
7	10k	DomainCategory	18
8	10k	KnowledgeDimension	4
9	10k	LearningDomain	3
10	10k	UniqueId	9

Table 7: Neo4j Database YDB10k

SL	Node(P)	Relationship(R)	Node(Q)	Count
1	AssessmentTechnique	ASSESSTECHHASDC	DomainCategory	10000
2	LearningObjective	ACTIONWORDCATEGORY	ActionWordCategory	10068
3	DomainCategory	DOMAIN CATEGORY	ActionWordCategory	58
4	LearningDomain	LDHASDC	DomainCategory	18
5	DomainCategory	LEARNING DOMAIN	LearningDomain	18
6	AssessmentTechnique	KNOWLEDGE DIMENSION	KnowledgeDimension	10000
7	DomainCategory	AWCHASDC	ActionWordCategory	58
8	AssessmentTechnique	ASSIGNED ASSESTECH	LearningObjective	9998
9	LearningObjective	CONTENTS	Content	10668
10	DomainCategory	LDHASDC	LearningDomain	18
11	KnowledgeDimension	KNOWLEDGE DIMENSION	AssessmentTechnique	10000
12	ImodUser	OWNS	Imod	10000
13	Content	HASCONTENT	Imod	10001
14	ActionWordCategory	AWCHASDC	DomainCategory	58
15	ActionWordCategory	ACTIONWORD	LearningObjective	10068

		CATEGORY		
16	AssessmentTechnique	ASSESSTECH HASLDOMAIN	LearningDomain	10000
17	Imod	HASCONTENT	Content	10001
18	Imod	IMOD	LearningObjective	10068
19	LearningDomain	LEARNING DOMAIN	DomainCategory	18
20	Content	CONTENTS	LearningObjective	10068
21	Imod	OWNS	ImodUser	10000
22	KnowledgeDimension	CONHASKD	Content	10001
23	DomainCategory	ASESSTECH HASDC	AssessmentTechnique	10000
24	LearningObjective	IMOD	Imod	10068
25	ActionWordCategory	DOMAIN CATEGORY	DomainCategory	58
26	Content	CONHASKD	KnowledgeDimension	10001
27	LearningObjective	PROVIDES	Imod	10068

Table 8: Neo4j Database YDB10k Relationships

SL No.	Dataset	Node Type	Count
1	20k	Imod	20000
2	20k	ImodUser	20000
3	20k	LearningObjective	20148
4	20k	Content	20001
5	20k	AssessmentTechnique	20000
6	20k	ActionWordCategory	59
7	20k	DomainCategory	18
8	20k	KnowledgeDimension	4
9	20k	LearningDomain	3
10	20k	UniqueId	9

Table 9: Neo4j Database YDB20k

SL No	Node(P)	Relationship(R)	Node(Q)	Count
1	AssessmentTechnique	ASSESSTECH HASDC	DomainCategory	20000
2	LearningObjective	ACTIONWORD CATEGORY	ActionWordCategory	20148
3	DomainCategory	DOMAIN CATEGORY	ActionWordCategory	58

4	LearningDomain	LDHASDC	DomainCategory	18
5	DomainCategory	LEARNING DOMAIN	LearningDomain	18
6	AssessmentTechnique	KNOWLEDGE DIMENSION	KnowledgeDimension	20000
7	DomainCategory	AWCHASDC	ActionWordCategory	58
8	AssessmentTechnique	ASSIGNED ASSESTECH	LearningObjective	20000
9	LearningObjective	CONTENTS	Content	19440
10	DomainCategory	LDHASDC	LearningDomain	18
11	KnowledgeDimension	KNOWLEDGE DIMENSION	AssessmentTechnique	20000
12	ImodUser	OWNS	Imod	20000
13	Content	HASCONTENT	Imod	20001
14	ActionWordCategory	AWCHASDC	DomainCategory	58
15	ActionWordCategory	ACTIONWORD CATEGORY	LearningObjective	20148
16	AssessmentTechnique	ASSESSTECH HASLDOMAIN	LearningDomain	20000
17	Imod	HASCONTENT	Content	20001
18	Imod	IMOD	LearningObjective	20148
19	LearningDomain	LEARNING DOMAIN	DomainCategory	18
20	Content	CONTENTS	LearningObjective	19440
21	Imod	OWNS	ImodUser	20000
22	KnowledgeDimension	CONHASKD	Content	20001
23	DomainCategory	ASESSTECH HASDC	AssessmentTechnique	20000
24	LearningObjective	IMOD	Imod	20148
25	ActionWordCategory	DOMAIN CATEGORY	DomainCategory	58
26	Content	CONHASKD	KnowledgeDimension	20001
27	LearningObjective	PROVIDES	Imod	20148

Table 10: Neo4j Database YDB20k Relationships

5.4 Challenges

While loading the data in to graph database Neo4j, one of the major challenges faced apart from memory constraints is to set a unique id for all the nodes created. All the traditional RDBMS like Oracle, PostgreSQL and MySQL comes with the feature of auto generating unique id for every row in a table. So, whenever a developer is creating an object and trying to save the object in the table with all the attributes, he does not need to worry about the uniqueness. But neo4j fails to do that. Neo4j does not have any tabular structure. It stores the data as a graph data model. It does generate unique ids but whenever we try to compact the database store, it might lose the ids generated against an object. There have been instances when Neo4j created ids have been re-used and developers have ended getting similar ids for two objects. These ids are non-incremental system generated and could not be used for assuring uniqueness property of the data model.

5.4.1 Alternatives/ Solutions:

In Table 11, pseudo code for creating a unique id for a Imod Object called Graph Database leveraging MERGE Command in Cypher is shown.

```
// get unique id
MERGE (id:UniqueId{name:'Imod'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
WITH id.count AS uid
```

```
// create Imod/Course node
CREATE (p:Imod{id:uid,imodName:'Graph Database',noOfSeats:60})
RETURN p AS Imod
```

Table 11: Psuedo code- UniqueId(1)

Next, another Imod Object with imodName as Computer Security has been created. The pseudo code is written in Table 12.

```
MERGE (id:UniqueId{name:'Imod'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
WITH id.count AS uid
// create Imod/Course node
CREATE (p:Imod{id:uid,imodName:'Computer Security',noOfSeats:60})
RETURN p AS Imod
```

Table 12: Psuedo code- UniqueId(2)

This time the ON CREATE line will not be executed as we already have that UniqueID singleton node. On retrieving all the Imod object, it is shown Figure 16 in that different imods have incremental ids as per the sequence of creation.

Figure 16: Output showing ids of different Imod Object

Similarly, unique ids for different domains can be generated leveraging the Singleton UniqueID node that we have created. For each of the domains, this UniqueID node will have different counters which is shown in Figure 17. Count for Imod is 2 as two Imod objects have been created.

Figure 17: UniqueId node

5.4.2 Explanation of Cypher MERGE Command

Merge acts as combination of MATCH and CREATE. It will try to find the pattern in the graph and if it does, nothing is created. If the pattern cannot be matched, only then will it be created. MERGE (u1:User {name: "u1"}) will try to find a User node with name=u1. If such a node cannot be found, it is created. Once created, re-executing this MERGE statement has no effect on the graph.

5.4.3 Performance Issue

Since we are calling the MERGE command every time we are creating an object and referring to that Singleton Node, so practically for each create statement we are running another query to get the count value of that node. It might have significant performance issues while dealing with large number of datasets. But to solve the problem of uniqueness, I think MERGE helps us a lot. Also, we can create non-numeric unique ids with MERGE command and use Neo4j indexes while querying thereby significantly reducing the query execution time for other cases.

CHAPTER 6

COURSE DESIGN COMPLETENESS

6.1 Progress Bar Feature

The progress bar feature allows the instructor to get a visual idea about the current status of the course design. The course design is based on completion of several key features. Upon successful implementation of every factor, the progress bar meter gradually reaches the course completion stage. The main features that determines the completion of a course design are enumerated below:

I. Course Overview Details- (Course Description, Schedule, Policy, Instructor etc.)
II. Learning Objectives
III. Content
IV. Assigned Pedagogy Techniques
V. Assigned Assessment Technique

Specific weightage has been given to each of the factors mentioned above. The progress meter algorithm works on the basic assumption that the activities are done in a pipelining fashion. It means that only after the completion of a few basic required details in the course overview tab, the user can move forward to the other tabs. The progress bar is dynamic in nature and re-calculates if any activity has been done on the existing IMOD. We have provided various color codes which effectively represents the completion stage of the course design.

6.2 Color Codes

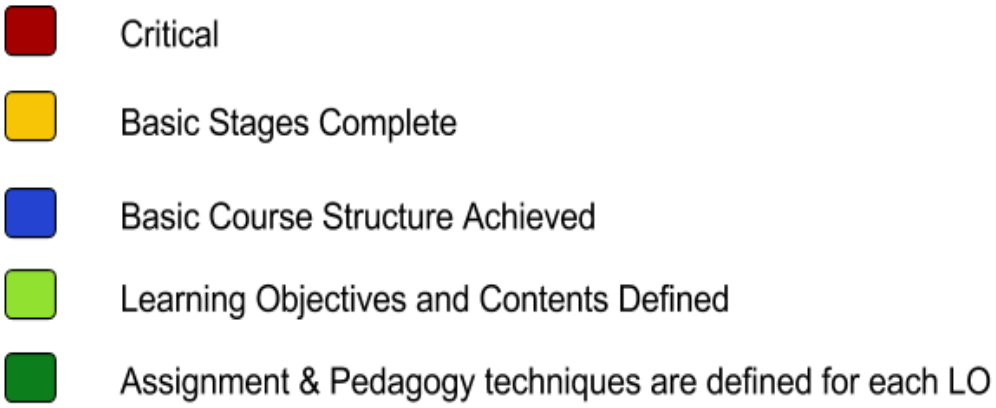

Figure 18: Color codes of Progress Bar

6.3 Algorithm of Completeness

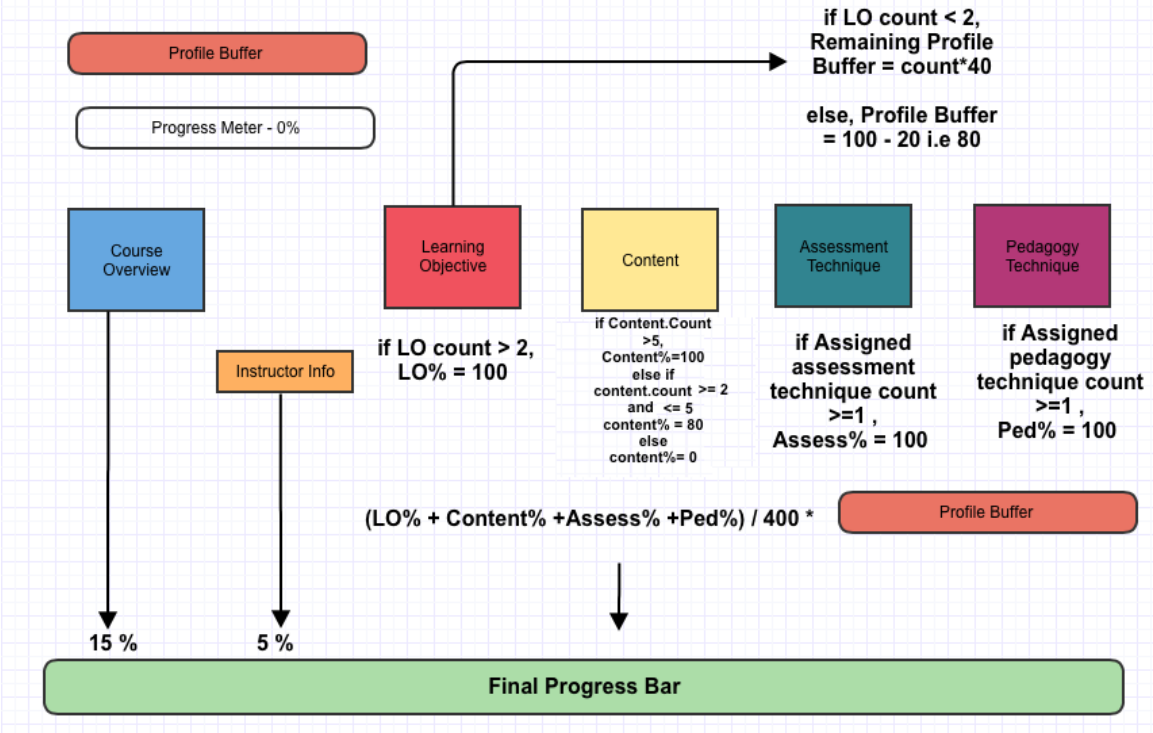

Figure 19: Graphical Picture of Progress Bar Algorithm

6.3.1 Course Completion Overview Percentage Allocation:

Initially, profile buffer is kept at 100 and profile percent at 0. If required fields completed, then 15% towards total profile percent is allocated. If instructor information is filled, then 5 % to the total calculation is allocated. The pseudo code for this calculation is presented in Table 13 below

```
isReqCODone = boolean flag for maintaining the status of required field inputs in
Course Overview Tool
isCDDone = boolean flag showing whether all required fields are filled or not
 isSchedComplete =  boolean flag
 isCDescGiven = boolean flag
 isCPolicyPresent = boolean flag
 isInstrDetailsFed = boolean flag


if(isCDDone && isSchedComplete  && isCDescGiven && isCPolicyPresent &&
isInstrDetailsFed):
        isReqCODone = true;
 else:
        isReqCODone = false;
```

Table 13: Psuedo code- Course Overview

6.3.2 Percentage Buffer Calculation

If required course overview details are fed, buffer is reduced to 80. Also, the actual profile buffer is determined by the count of learning objectives(LO). The pseudo code for this calculation is presented in Table 14 below.

```
isReqCODone:
        percentageBuffer = 100-20
int getLONum() = returns number of learning objective added.
if getLONum() > 2:
        Allocate percentage buffer = 80
else:
        allocate percentageBuffer = getLONum() * 40
```

Table 14: Psuedo code- Percentage Buffer

6.3.3 Learning Objective (LO) Completion Calculation

If learning objective count is greater than 2, LO percentage is allocated to be100% of the predefined profile buffer. The pseudo code for this calculation is presented in Table 15 below.

```
isPerfPresent = boolean flag showing if all mandatory fields are filled
isContentpresent = boolean flag
```

```
isConditionGiven = boolean flag

isCriteriaGiven = boolean flag

LO percentage = 0

If all of the variables above are true & LO count > 2:

        Return LO percentage as 100
```

Table 15: Psuedo code- Learning Objective

6.3.4 Content Percent Calculation

If number of contents added to an imod course is less than 5, then 80% of content percentage is added. Otherwise, full 100% of allocated buffer is added. The pseudo code for this calculation is presented in Table 16 below.

```
If getContentCount () > 5:

        Return content Percent as 100

 else if getContentCount() >= 2:

        Return ContentPercent as 80

else:

        Return ContentPercent as 0
```

Table 16: Psuedo code- Content

6.3.5 Assessment Percentage Calculation

If assessment techniques are assigned to an imod, 100% assessment percentage is allocated. The pseudo code for this calculation is presented in Table 17 below.

```
int getAssignedTechniques = returns number of assigned techniques to imod
if getAssignedTechniques > 0
        Return assessment percentage as 100
else :
        Return assessment percentage as 0
```

Table 17: Psuedo code- Assessment Technique

6.3.6 Pedagogy Percent Calculation

If pedagogy techniques are assigned to an imod, 100% pedagogy percentage is allocated. The pseudo code for this calculation is presented in Table 18 below.

```
int getAssignedTechniques = returns number of assigned techniques to imod
if getAssignedTechniques > 0
        Return pedagogy percentage as 100
else :
        Return pedagogy percentage as 0
```

Table 18: Psuedo code- Pedagogy Technique

6.3.7 Total Calculation

After calculating the required percentage of each of the components using different queries, total profile percent is calculated by adding proportionate weight from

the profile buffer towards the calculation of total profile percent. The pseudo code for this calculation is presented in Table 19 below.

```
profileBuffer = 100
profilePercent = 0
if(ReqCODone):
        if(instructorDetailsFed)
                profilePercent = 20
                profileBuffer -= 20
        Else:
                profilePercent = 15
                profileBuffer -=20

profilePercent += (getLOpercent + getAssessmentPercent + getContentPercent +
getPedagogyPercentage )/400 * profilebuffer
Return profilePercent
```

Table 19: Psuedo code- Final Progress Percentage Calculation

6.3.8 Stages of Course Design

This section describes in detail about the various stages of course design. Initially, while creating a new instructional module (IMOD), no details are saved. Hence, as per the algorithm depicted in Figure 19, the course design completion percentage will be 0% as shown in Figure 20.

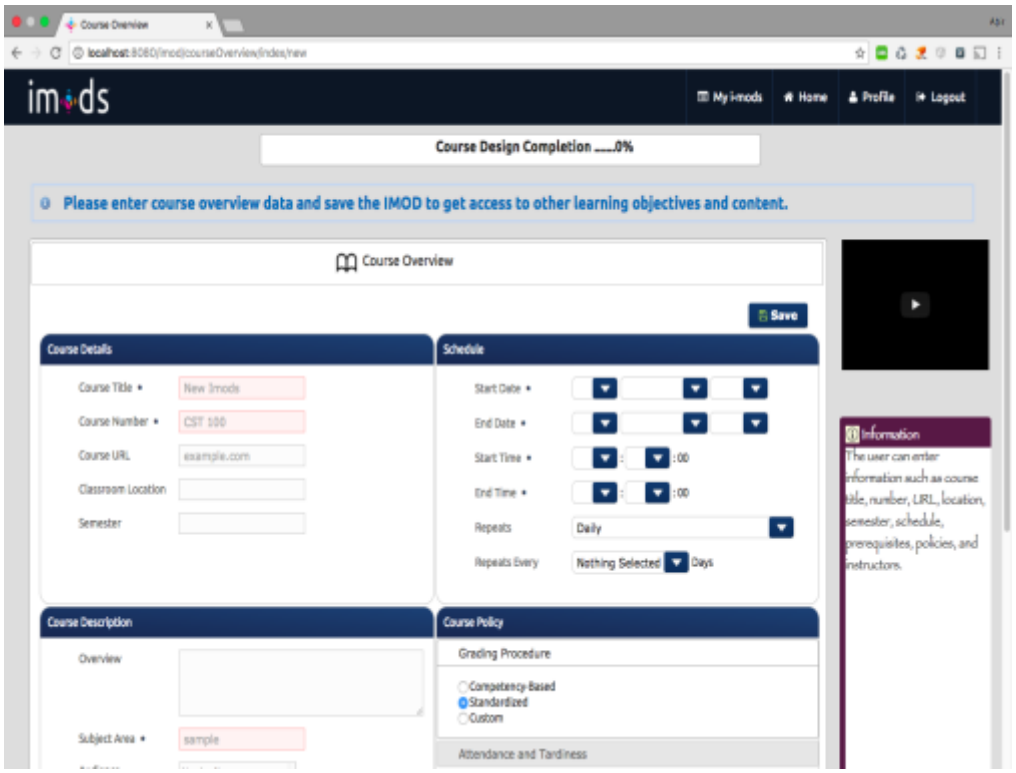

Figure 20: IMODS Progress Bar-Stage 0

In the next stage, as the instructor fed in all the course overview details like course number, title, course policy, course description etc., as per the algorithm in Figure 19, the percentage reaches up to 15% as shown in Figure 21 in stage 1 of course design

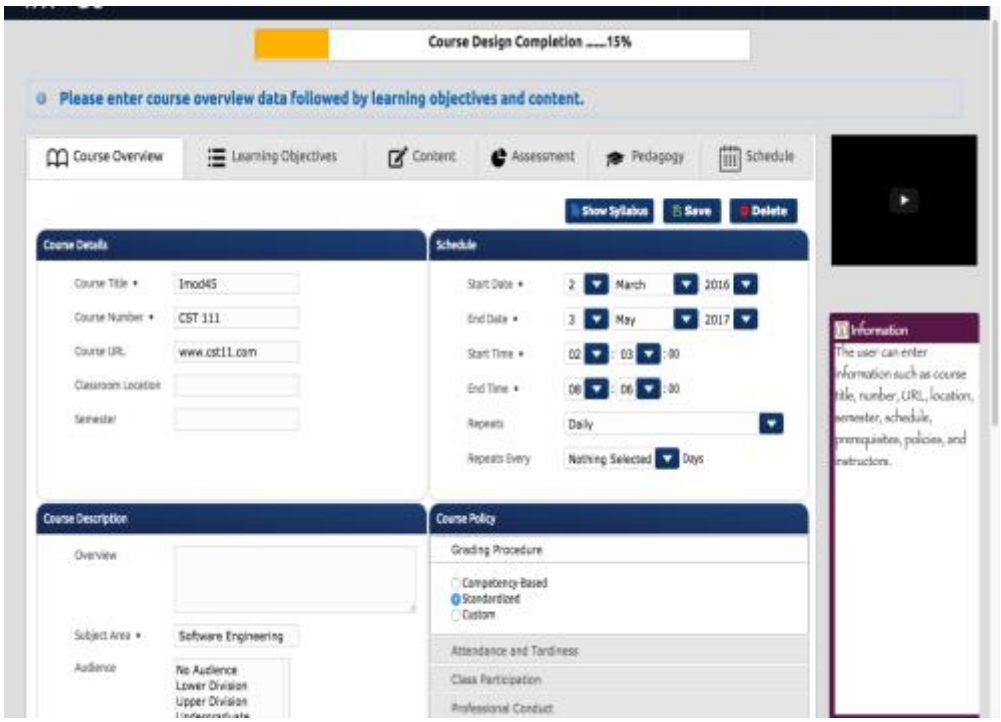

Figure 21: IMODS Progress Bar-Stage 1

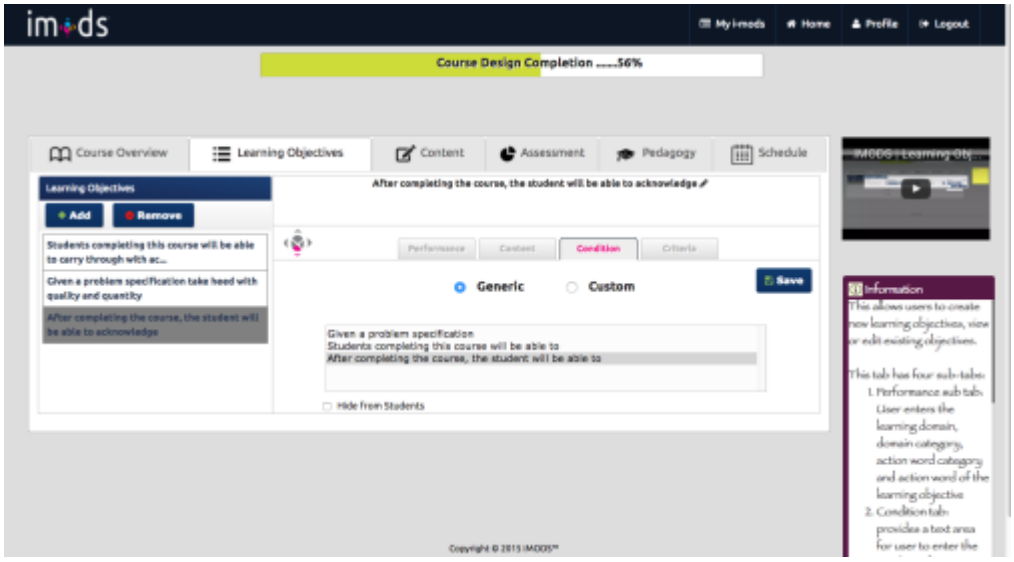

Figure 22: IMODS Progress Bar-Stage 2

In the above Figure 22, the stage 2 of course design process is shown. Learning objectives are being created in this stage by systematically adding performance, condition and criteria features of learning objectives. If less than 2 learning objectives has been created, remaining profile buffer will be learning objective count* 20. Otherwise, the profile buffer will be 100-(course overview + instructor information) i.e. 100 -15 -5 = 80.

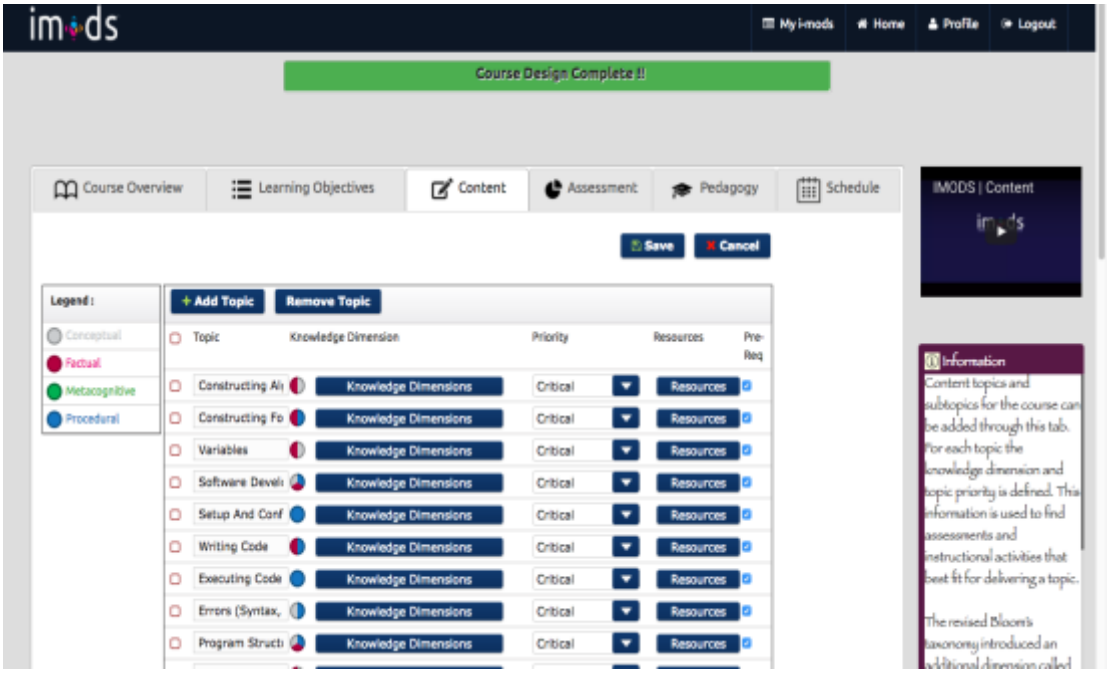

Figure 23: IMODS Progress Bar-Stage 3

In stage 3, content or topics are being created and added to the instructional module(imod) shown in Figure 23. If the number of content exceeds 5, the total percentage of content share is added to the final calculation is 100% of the weighted share of content in profile buffer. If the number remains between 2 and 5, 80% of weighted share in profile buffer is added to the total calculation.

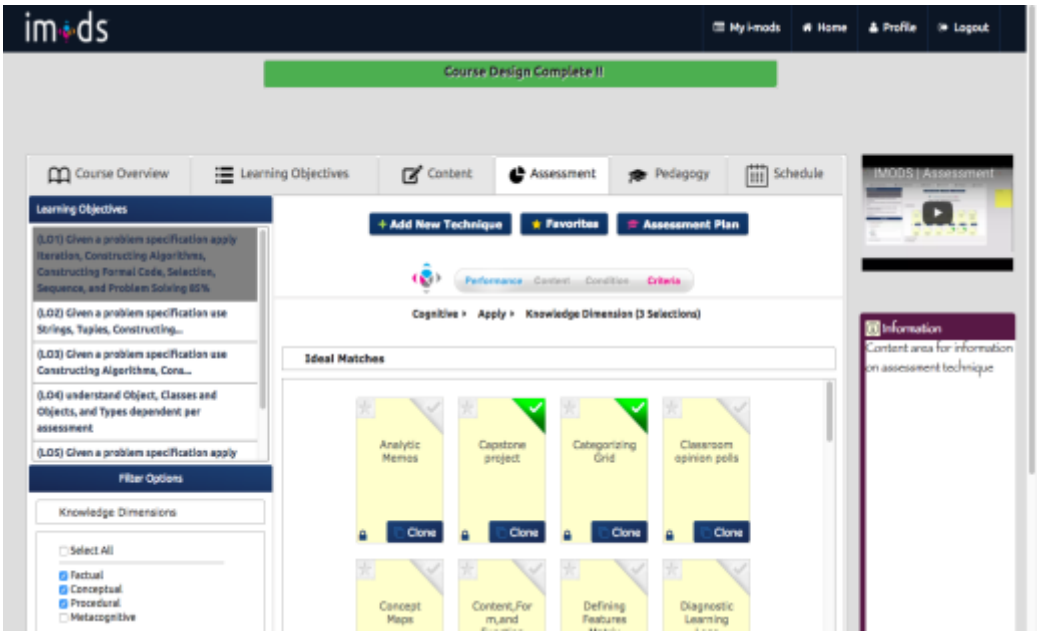

Figure 24: IMODS Progress Bar-Stage 4

In stage 4, assessment techniques are created and being assigned to particular learning objectives on the left-side column in Figure 24. Instructional modules are being assessed against these strategies and the ultimate performance measure of the student is dependent on these strategies. Assigned of a strategy in a learning objective contributes 100% of the weighted share in the final calculation. Similarly, pedagogy techniques are also created and assigned to learning objectives as well. This summarizes the entire instructional module design process.

6.3.9 Correctness of Course Design Completion Calculation Algorithm

In this section, experiments are conducted for proving the correctness of the algorithm. 10 instructional modules or courses are created with different degree of completeness. Queries are fired dynamically after each stage of course design to calculate the final measure of course design completion. In the first stage, each of the newly created instructional modules are provided course overview details and instructor's information (column 2 in Table 20). Then different learning objectives are created for this instructional module based on PC^3 framework [6]. In this experiment, queries are fired to calculate the number of learning objectives created for a course and based on the count (column 3 in Table 20), percentage towards final completion measure is allocated. Then contents are created for an instructional module and added to each of the learning objectives. Dynamic queries fetching the total number of content (column 4 in Table 20) also contributes towards the final measure. After this stage, assessment strategies and pedagogy techniques are created and assigned to each learning objective. Based on the total number of assigned assessment strategies (column 5) and pedagogy techniques (column 6), percentage towards the final measure of course design is allocated.

In Table 20, a comparative scenario is presented to show the actual completion percentage vs the calculated completion percentage by the algorithm for different instructional modules at different stages of completion. Comparisons are performed between Actual Completion (column 7) and Calculated Completion (column 8) and it is evident that for every instructional module at different stages of completion, the values of these two columns are same. This proves the correctness of the algorithm. This algorithm

helps the instructors in visualizing their course design progress and hence improves the usability of the Instructional Module Development System (IMODS) [21].

SL No.	Course & Instructor Info	Learning Objective Count	Content / Topic Total Count	Assigned Assessment Technique Total Count	Assigned Pedagogy Technique Total Count	Actual Completion (%)	Calculated Completion (%)
1	YES	1	1	6	5	40	40
2	YES	10	4	3	3	96	96
3	YES	6	6	5	3	100	100
4	NO	0	0	0	0	0	0
5	YES	1	4	6	8	48	48
6	YES	3	6	2	2	100	100
7	YES	1	6	7	6	50	50
8	YES	2	4	0	1	56	56
9	YES	1	7	0	0	30	30
10	YES	3	2	1	0	76	76

Table 20: Completion Measure of Instructional Modules –Evaluation Results

CHAPTER 7

EXPERIMENTS FOR COMPARATIVE ANALYSIS OF DATABASES

7.1.1 Experiment Setup

As per the research statement, the first research question is to evaluate the performance of graph database (Neo4j) and relational database (PostgreSQL) on Instructional Module Development System based on the response time of different complex queries. For the purpose of experiment, three different instances of Neo4j v3.2.1 databases YDB5k, YDB10k and YDB10k have been created. Similarly, three different PostgreSQL v9.4 database instances xDB5k, xDB10k & xDB20k have been created. These databases have been loaded with corresponding 5k,10k and 20k datasets. Two different instances of IMODS application has been implemented- one with PostgreSQL as primary data store called ProjectX and another with Neo4j as primary data store called ProjectY. These applications have been developed using Grails 3.2.2 framework and implemented abstracting the domain structure of IMODS.

7.1 Machine Configuration

For the purpose of this experiment, a Mac machine with 1.6 GHz Intel Core i5 processor, 4 GB 1600 MHz DDR3 RAM memory and 128 GB Hard Disk Space is used. The operating system installed in the machine was OS X El Capitan v10.11.2. For developing the IMODS application, Grails 3.2.2 framework has been installed. Neo4j

v3.2.2 as well as PostgreSQL v9.4 servers have been installed. pgAdmin client has been used to interact with PostgreSQL.

7.2 Experiment

In this experiment, we have run several queries as listed in section 7.2.1 on both Neo4j server and PostgreSQL server and compared the mean response time. These queries have varying degree and depth of relationships in them. Before actually, looking at the queries, let us look at the graph database statistics for different datasets.

7.2.1 Queries

Seven different queries which runs through different stages of course design in IMODS have been evaluated. Each of the queries have been fired 5 times for each database and finally the mean response time has been counted. Multiple times execution has been carried out to remove any kind of caching effect or any other biased behavior of the system. Care has been taken that the system will have no other process running while executing these queries so that maximum CPU memory is available. No performance tuning or database tuning has been done to either of Neo4j and PostgreSQL servers to maintain transparency and fairness to the experiments. In PostgreSQL, these queries were implemented using SQL while in Neo4j, these queries were implemented with Cypher. Below each of the queries are listed with their Cypher and SQL equivalent code.

1. Find all Imods with learning objective's (LO) that do not have at least 1 assessment assigned.

   Cypher:

   Match (lo:LearningObjective) where NOT (lo)-[:assignedLearningObjective]->() return distinct lo.imod

   SQL:

   Select distinct(lo.imod_id) from learning_objective lo LEFT OUTER JOIN learning_objective_assessment_techniques at ON lo.id = at.learning_objective_id where at.assessment_technique_id IS NULL ORDER BY lo.imod_id

2. For a given IMOD, identify learning objective's (LO) whose assigned assessments are not consistent with LO's LearningDomain(LD) i.e. LO.LD != Assigned Assessment.LearningDomain(LD)

   Cypher:

   Match(ld:LearningDomain)<-[:ASSESSTECHHASLDOMAIN]-(at:AssessmentTechnique)<-[:assignedLearningObjective]-(lo:LearningObjective)-[:actionWordCategory]->(awc:ActionWordCategory)-[:AWCHASDC]->()-[:LEARNINGDOMAIN]->(ldom:LearningDomain) where lo.imod=513 and ld.__id__ <> ldom.__id__ Return lo.imod,lo.__id__,lo.awc,ldom.__id__,at.__id__,ld.__id__ ORDER BY lo.__id__

   SQL:

Select A.imod_id AS Imod,A.id AS LO,A.action_word_category_id AS AWC,E.learning_domain_id,B.assessment_technique_id AS AssesTech,C.learning_domain_id from learning_objective A INNER JOIN learning_objective_assessment_techniques B ON A.id = B.learning_objective_id INNER JOIN assessment_technique_learning_domain C ON B.assessment_technique_id = C.assessment_technique_learning_domain_id INNER JOIN action_word_category D ON A.action_word_category_id = D.id INNER JOIN domain_category E ON D.domain_category_id = E.id where A.imod_id = 513 and C.learning_domain_id != E.learning_domain_id

3. For a given IMOD, identify learning objective's(LO) whose assigned assessments are not consistent with LO's domain category(DC) i.e. LO.DC != Assigned Assessment.DC

   Cypher:

   Match(dc:DomainCategory)<-[:ASSESSTECHHASDC]-(at:AssessmentTechnique)<-[:assignedLearningObjective]-(lo:LearningObjective)-[:actionWordCategory]->(awc:ActionWordCategory) where lo.imod=513 and dc.__id__ <> awc.dc Return lo.imod,lo.__id__,lo.awc,awc.dc,at.__id__,dc.__id__ ORDER BY lo.__id__

SQL:

Select A.imod_id AS Imod,A.id AS LO,A.action_word_category_id AS AWC,D.domain_category_id,B.assessment_technique_id AS AssesTech,C.domain_category_id from learning_objective A INNER JOIN learning_objective_assessment_techniques B ON A.id = B.learning_objective_id INNER JOIN assessment_technique_domain_category C ON B.assessment_technique_id = C.assessment_technique_domain_category_id INNER JOIN action_word_category D ON A.action_word_category_id = D.id where A.imod_id = 513 and C.domain_category_id != D.domain_category_id

4. For a given IMOD, identify LO's whose assigned assessments are not consistent with LO's content's knowledge dimension(KD) i.e. LO.Content.KD != Assigned Assessment.KD

Cypher:

Match(kd:KnowledgeDimension)<-[:KNOWLEDGEDIMENSION]-(at:AssessmentTechnique)<-[:assignedLearningObjective]-(lo:LearningObjective)-[:CONTENTS]->(con:Content)-[:conHasKD]->(conKd:KnowledgeDimension) where lo.imod=513 and kd.__id__ <>

conKd.__id__ Return

lo.imod,lo.__id__,con.__id__,conKd.__id__,at.__id__,kd.__id__ ORDER BY

lo.__id__

SQL:

Select A.imod_id AS Imod,A.id AS LO,D.content_id AS

Content,E.knowledge_dimension_id,B.assessment_technique_id AS

AssesTech,C.knowledge_dimension_id from

learning_objective A INNER JOIN learning_objective_assessment_techniques B

ON A.id = B.learning_objective_id

INNER JOIN assessment_technique_knowledge_dimension C ON

B.assessment_technique_id = C.assessment_technique_knowledge_dimension_id

INNER JOIN learning_objective_contents D ON A.id = D.learning_objective_id

INNER JOIN content_knowledge_dimension E ON D.content_id =

E.content_content_dimensions_id

where A.imod_id = 513 and C.knowledge_dimension_id !=

E.knowledge_dimension_id Order By A.id

5. Identify all LO's with content having Critical priority and no assessment technique assigned.

Cypher:

Match (at:AssessmentTechnique)<-[:assignedLearningObjective]-(lo:LearningObjective)-[:CONTENTS]->(con:Content{priority:'Critical'}) WITH count(at.__id__) as CNT,lo where CNT = 0 return lo.__id__, CNT

SQL:

Select distinct loat.learning_objective_id, count(loat.assessment_technique_id) from learning_objective_assessment_techniques loat INNER JOIN learning_objective_contents loc ON loat.learning_objective_id = loc.learning_objective_id INNER JOIN content con ON con.id = loc.content_id where con.priority='Critical' GROUP BY loat.learning_objective_id HAVING count(loat.assessment_technique_id) = 0

6. Identify all LO's with content having 'Critical' priority and less than 2 assessment technique assigned.

Cypher:

Match (at:AssessmentTechnique)<-[:assignedLearningObjective]-(lo:LearningObjective)-[:CONTENTS]->(con:Content{priority:'Critical'}) WITH count(at.__id__) as CNT,lo where CNT < 2 return lo.__id__, CNT

SQL:

Select distinct loat.learning_objective_id, count(loat.assessment_technique_id) from learning_objective_assessment_techniques loat INNER JOIN learning_objective_contents loc ON loat.learning_objective_id = loc.learning_objective_id INNER JOIN content con ON con.id = loc.content_id where con.priority='Critical' GROUP BY loat.learning_objective_id HAVING count(loat.assessment_technique_id) < 2

7. For a given IMOD, identify all 'In-class' assessments.

   Cypher:

   Match(lo:LearningObjective{imod:513})-[:assignedLearningObjective]->(at:AssessmentTechnique) where at.whereToCarryOut = 'In-class' Return at.__id__

   SQL:

   *Select * from assessment_technique ast INNER JOIN learning_objective_assessment_techniques loat ON ast.id = loat.assessment_technique_id INNER JOIN learning_objective lo ON lo.id = loat.learning_objective_id where ast.where_to_carry_out = 'In-class' and lo.imod_id = 513*

The above listed queries are fired at different stages of course design in Instructional Module Development System(IMODS) [21]. These queries help in guiding the instructor to find out inconsistencies in relationships among various course components. The queries are designed such that they are useful to evaluate of performance of graph database and relational database. Queries 1 will help to find out all the instructional modules that do not have any assessment strategy assigned to their learning objectives. Queries 2, 3 and 4 helps to find out all erroneous instructional modules designed that have inconsistent learning domains between learning objectives and assigned assessment strategies, inconsistent knowledge dimensions of contents and assigned assessment techniques etc. Query 5 finds out all learning objectives with critical priority contents and no assessments assigned. This query checks how databases behave with NULL comparisons. Similarly, query 6 finds all critical priority content with less than two assessments. This query evaluates the numeric value matching performance of both Neo4j and PostgreSQL. Query 7 finds all assessments that can be conducted in class and is a perfect query for evaluating string matching. These queries help the instructor to design the course effectively with highly connected and tightly aligned components as per PC^3 framework [6]. In the next chapter, results of response time for each of the above queries have been compared for both Neo4j and PostgreSQL. Also, effort has been made to analyze the different query response time as the dataset grows and the possible reasons for such results.

CHAPTER 8

RESULTS & CONCLUSION

8.1 Experiment Results

In this section, the results of the experiments obtained from our experiments performed on Neo4j and PostgreSQL are shown. The mean response time for all the seven queries by Neo4j and PostgreSQL for each of the 5k, 10k and 20k dataset are considered. Each of the queries is fired 5 times to avoid any effect of caching and to prevent the experiments from any bias. Columns starting from N1 to N5 represents the response time for all the 5 times whenever a query is fired in Neo4j. Similarly, columns starting from P1 to P5 represents the response time for all the 5 times whenever a query is fired in PostgreSQL. Columns Mean_neo4j and Mean_postgres represents the mean response time of each query.

SL No.	**N1**	**N2**	**N3**	**N4**	**N5**	**Mean_ neo4j (in ms)**	**P1**	**P2**	**P3**	**P4**	**P5**	**Mean_ postgres (in ms)**
1	37	29	44	18	26	30.8	37	27	31	28	37	32
2	44	58	32	56	39	46.6	45	34	34	28	18	31.8
3	52	34	24	20	26	31.2	33	38	24	20	22	27.4
4	50	48	34	39	42	42.6	40	37	36	32	38	36.6
5	43	38	47	44	39	42.2	33	31	33	30	29	31.2
6	66	78	59	72	68	68.6	34	33	30	31	34	32.4
7	27	21	28	23	19	23.6	19	20	13	11	16	15.8

Table 21: Response Time for 5k Imod User Dataset

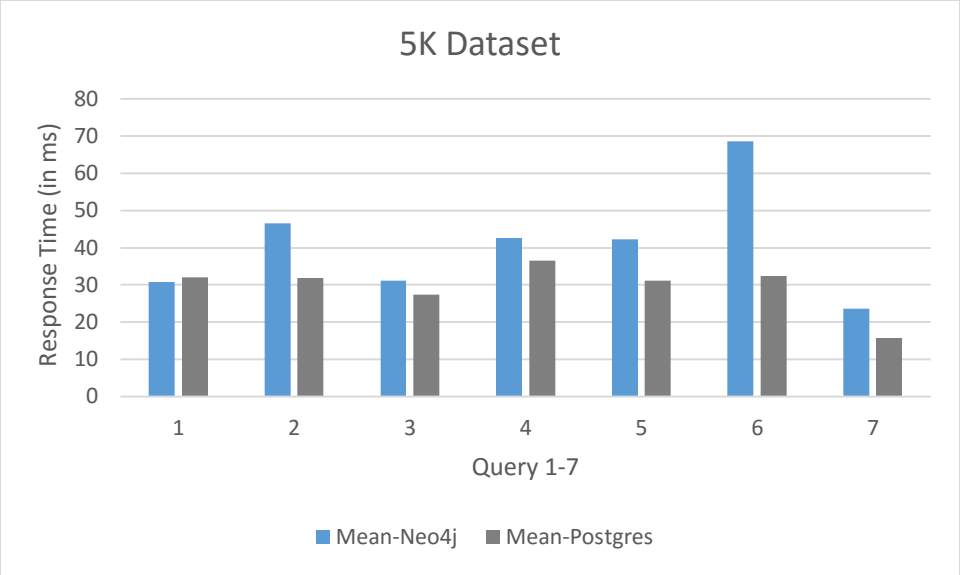

Figure 25: Comparison of Neo4j and PostgreSQL Mean Response Time (5k dataset)

After the completion of experiments on 5k dataset, it has been concluded from the results shown in Figure 25 that for most of the queries PostgreSQL performance is better than Neo4j. However, for queries 3 and 4 the mean response time in Table 21 is similar for both PostgreSQL and Neo4j. On inspecting the query 3 and 4 structure mentioned in Chapter 7.2.1, it has been found that these queries have 3 levels of connections and their performance might vary by increasing the size of the dataset. Based on this information, it has been decided the next round of experiments should be conducted with at least 10k dataset.

SL No.	N1	N2	N3	N4	N5	Mean_ neo4j (in ms)	P1	P2	P3	P4	P5	Mean_ postgres (in ms)
1	77	78	70	58	52	67	48	45	60	57	62	54.4
2	46	64	58	44	42	50.8	48	50	45	37	52	46.4
3	15	12	7	8	10	10.4	44	35	30	37	28	34.8
4	16	12	8	10	16	12.4	66	57	38	48	52	52.2
5	90	78	88	82	66	80.8	68	56	55	76	70	65
6	87	100	78	88	72	85	67	56	56	51	66	59.2
7	37	28	32	25	29	30.2	44	28	32	32	34	34

Table 22: Response Time for 10k Imod User Dataset

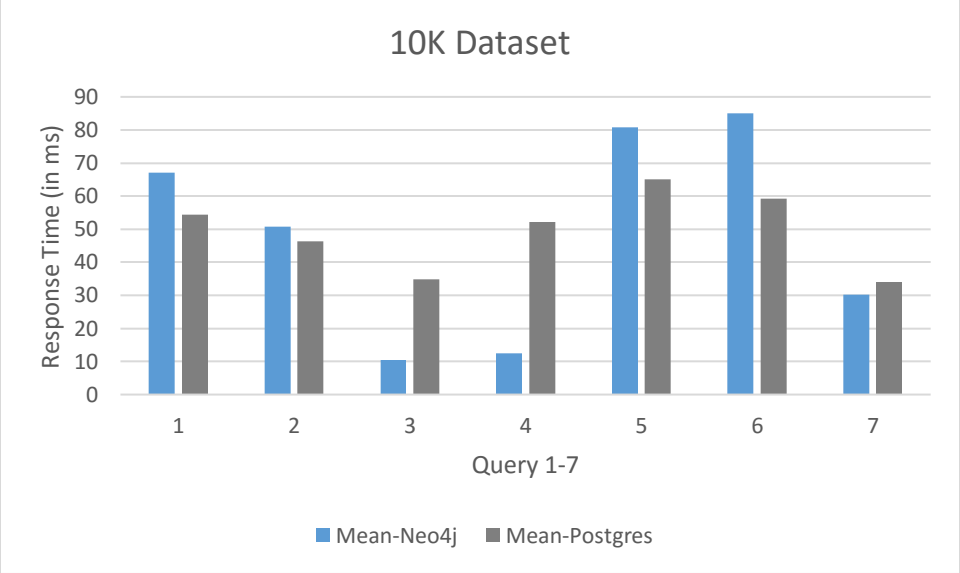

Figure 26: Comparison of Neo4j and PostgreSQL Mean Response Time (10k dataset)

After completion of experiments in this phase with 10k dataset, there has been improvement in query performance in query 3 and query 4 as shown in Figure 26 which can be attributed to the fact that with increase in size of dataset, pre-materialized

relationships in Neo4j contributed in faster access than PostgreSQL. But even then, query 5 and query 6 having NULL comparison and numeric value matching patterns performances have not improved from the last experiment. To justify their behavior properly, it has been decided to conduct another round of experiments further increasing the dataset size to 20k.

SL No.	N1	N2	N3	N4	N5	Mean_ neo4j (in ms)	P1	P2	P3	P4	P5	Mean_ postgres (in ms)
1	87	88	98	85	71	85.8	83	81	82	83	83	82.4
2	40	45	50	44	48	45.4	37	54	54	71	40	51.2
3	24	24	20	24	18	22	66	36	54	55	55	53.2
4	9	6	8	9	6	7.6	49	50	32	44	48	44.6
5	128	96	86	87	88	97	115	86	66	85	88	88
6	109	98	110	107	104	105.6	117	116	99	117	116	113
7	10	8	7	9	8	8.4	31	32	38	36	38	35

Table 23: Response Time for 20k Imod User Dataset

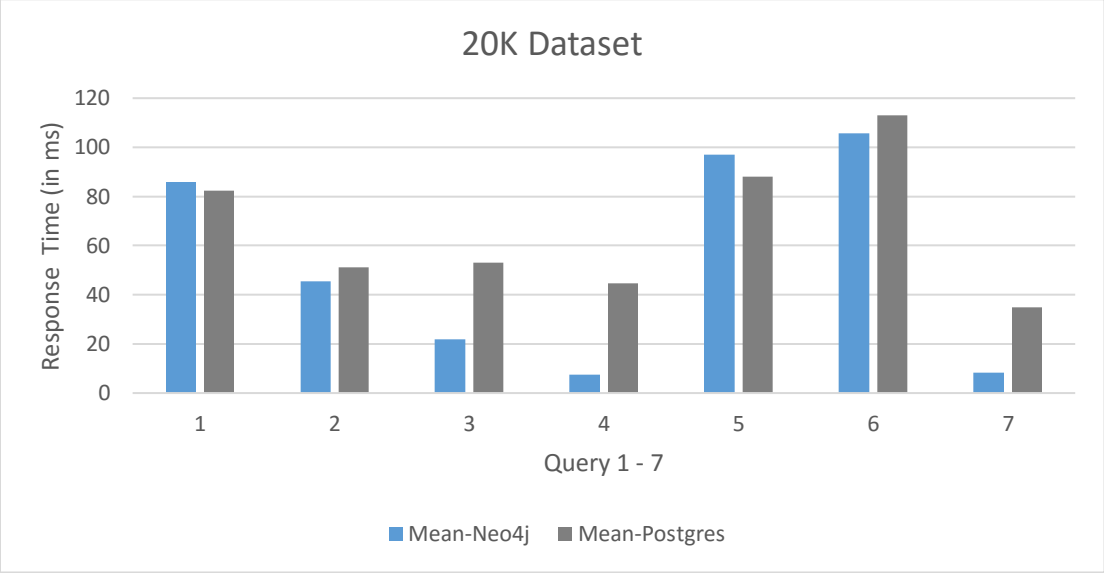

Figure 27: Comparison of Neo4j and PostgreSQL Mean Response Time (20k dataset)

After conducting this experiment with 20k dataset, the performance of query 2, 3, 4 and 7 have improved in Neo4j as shown in Figure 27. The initial hypothesis has been proven correct as performance of queries with greater number of connections and traversals have improved. Query 7 which involves string matching also improved as Neo4j uses Lucene based indexing [2] which is optimized for string which makes string value searches faster in Neo4j and hence the improvement is justified.

8.2 Analysis

The analysis of the results brought out key insights to the experiments conducted which are enumerated below:

i) From the above Table 21, we can observe that in 5k dataset for most of the queries the mean response time of PostgreSQL is better or similar than Neo4j. If the queries are similar with lesser complexity, the response time is similar or close like query 1. But if the query becomes complex having greater depth of relationships like query 3 and query 4, the performance varies. For smaller 5k dataset, the performance of PostgreSQL is better as look up operation is not much because of less number of rows and with indexing it becomes even more faster.

ii) From Table 22, it can be observed that in 10k dataset for query 3 and query 4 the performance of Neo4j improves exceptionally. The mean response time is more than 50% lower than the response time of PostgreSQL. It can be attributed to the fact that query 3 and query 4 involves traversing up to 3 to 4 level of connection depths. In these queries, the key focus is to find out inconsistencies in different instructional modules where the learning objective domain category is not like the assigned assessment

technique domain category and to find inconsistencies where content's knowledge dimension is different from assigned assessment technique's knowledge dimension. For query 7 which is a string matching query, the performance of Neo4j is almost 4 times better than PostgreSQL.

iii) From Table 23, it is observed that for 20k dataset, the performance of Neo4j further improves and the mean response time is much lower than PostgreSQL. For query 3 and 4, the mean response time is 50 to 70% better than PostgreSQL and same for query 7. But for query 5 and 6, even with large dataset of 20k, the performance of PostgreSQL is better than Neo4j. It makes it a better choice if we are dealing with larger dataset in our application with deeply connected nodes to choose Neo4j over PostgreSQL.

iv) From Table 22 & 23, we can observe that for query 5 which matches all learning objectives with Critical content that do not have any assignment, the performance of Neo4j is not that great even with larger dataset like 10k and 20k in size. Neo4j cypher queries struggle with NULL comparisons.

v) From the above tables, for query 6, which involves numeric value matching, the performance of Neo4j is not good in all the datasets. But for query 7 which involves string matching the performance of Neo4j is exceptionally better than PostgreSQL.

Since Neo4j explicitly stores relationships, they pre-materialize all relationships in to database structures. With indexed attributes, graph database performance increases order of several magnitudes in JOIN heavy queries because of this relationship pre-materializing ability. Neo4j using the index free adjacency graph processing technique avoids the need of lookup and directly hops onto connected edges to find the target nodes [2].

8.3 Conclusion & Future Work

In this thesis, the performance of Neo4j and PostgreSQL in IMODS have been evaluated for 7 different queries each requiring 1 to 4 levels of JOIN operations for traversing relationships between connected course components and also involving matching of string literals and numeric values. It has been observed that for an instructional course design application like IMODS, Neo4j is a good option when the dataset reaches 10k size or more. For any smaller dataset, PostgreSQL with its robust ACID conformance is more useful. The comparison between the two database servers encompassed 6 different databases and three data size configurations. For most of the seven queries in 5k dataset, performance of PostgreSQL and Neo4j is similar. But for datasets of size 10k and more, Neo4j outperforms PostgreSQL which involves 3 or 4 level of JOIN operations (in relational model). For IMODS, we can expect greater data sizes as number of courses will increase and greater depth of relationships will be added. Neo4j can be effective under those scenarios where key insights need to be retrieved to infer valuable information so instructors can make their course design more effective and intuitive. Frequent graph traversal operations need to be performed that would make the incorporation of Neo4j as the primary data store justified. One of the key limitations of this research work is that all the comparisons have been made on a single server and not in a distributed environment. Comparison of graph database and relational database on a distributed environment can be conducted as future work of this thesis. If data relationships stored in Neo4j servers are scattered geographically, it would take a toll on the performance due to network latency. This can be a great topic for research that will

help us in better understanding the effectiveness of graph database. Another area which can be a potential future work is development of an automated tool that can translate all foreign key relationships found in intermediate join tables in relational databases in to explicit data relationships in graph database. From the context of IMODS application, future research can be carried out in designing an alert system or a feedback mechanism to instructors which would help them to visualize all the inconsistencies present in their course design and where immediate action is required based on priority to achieve completeness in course design.

REFERENCES

[1] E. F. Codd, "Relational database: a practical foundation for productivity," *Communications of the ACM,* vol. 25, no. 2, pp. 109-117 , February 1982.

[2] M. Hunger, R. Boyd and W. Lyon, "The Definitive Guide to Graph Databases for the RDBMS Developer," Neo4j, 2016. [Online]. Available: https://info.neo4j.com/rs/773-GON-065/images/Definitive-Guide-Graph-Databases-for-RDBMS-Developer.pdf. [Accessed 2017].

[3] R. Barker, "Relational is not enough," in *Relational Databases: State of the Art Report 14:5*, D. A. Bell, Ed., 1986, pp. 15-23.

[4] S. Bansal, O. Dalrymple and A. Gaffar, "Design, Development, and Implementation of the Instructional Module Development System (IMODS)," in *American Society for Engineering Education Conference(ASEE)*, Seattle, 2015.

[5] S. Bansal, A. Gaffar and O. Dalrymple, "Building Faculty Expertise in Outcome-based Education Curriculum Design," in *Frontiers in Education Conference (FIE)*, El Paso,TX,USA, 2015.

[6] K. Andhare, O. Dalrymple and S. Bansal, "Learning Objectives Feature for the Instructional Module Development System," in *Proceedings of the 2012 ASEE PSW Section Conference*, San Luis Obispo.

[7] S. Bansal and O. Dalrymple, "Instructional Module Development System (IMODS)," in *21st ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, Peru, 2016.

[8] R. Mager, Preparing Instructional Objectives, 2nd ed., Belmont, CA: David S. Lake Publishers, 1984.

[9] D. Clark, "Bloom's Taxonomy of Learning Domains," 1999. [Online]. Available: http://www.nwlink.com/~donclark/hrd/bloom.html. [Accessed July 2017].

[10] A. LaMotte, "An Introduction to Bloom's Taxonomy for Instructional Designers," 2017. [Online]. Available: https://community.articulate.com/articles/blooms-taxonomy-elearning-instructional-design. [Accessed July 2017].

[11] O. Dalrymple and S. Bansal, "Repository of Instructional and Assessment Techniques for OBE-based Instructional Module Development system," *Journal of Engineering Education Transformations,* 2015.

[12] A. Martinez, R. Mora, D. Alvarado, G. L`opez and S. Quir`os, "A Comparison between a Relational Database and a Graph Database in the context of a Personalized Cancer Treatment Application," in *CEUR Workshop Proceedings*, 2016.

[13] F. Holzschuher and R. Peinl, "Performance of graph query languages: Comparison of cypher, gremlin and native access in Neo4j," in *Joint EDBT/ICDT 2013 Workshop GraphQ*, Genoa, Italy, 2013.

[14] T. Heath, "Linked Data - Connect Distributed Data across the Web," 2009. [Online]. Available: http://linkeddata.org/faq. [Accessed July 2017].

[15] F. Cerbah, "Learning highly structured semantic repositories from relational databases," in *European Semantic Web Conference*, 2008.

[16] J. F. Sequeda, M. Arenas and D. P. Miranker, "On directly mapping relational databases to rdf and owl," in *Proceedings of the 21st international conference on World Wide Web*, Lyon, france, 2012.

[17] W. Hu and Y. Qu, "Discovering simple mappings between relational database schemas and ontologies.," in *Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference*, 2007.

[18] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. CHen and D. Wilkins, "A Comparison of a Graph Database and a Relational Database - A Data Provenance Perspective," in *48th Annual Southeast Regional Conference (ACM SE '10)*, Oxford, Mississippi, 2010.

[19] S. Batra and C. Tyagi, "Comparative analysis of relational and graph databases," *International Journal of Soft Computing and Engineering(IJSCE)*, vol. 2, no. 2, May 2012.

[20] G. Adorni, S. Battigelli, D. Brondo, N. Capuano, M. Coccoli, S. Miranda, F. Orciuoli, L. Stanganelli, A. M. Sugliano and . G. Vivanet, "CADDIE and IWT: two different ontology-based approaches to Anytime, Anywhere and Anybody Learning," *Journal of e- Learning and Knowledge Society-English Version*, vol. 6, no. 2, 2010.

[21] IMOD System, "IMOD Framework," [Online]. Available: http://imod.poly.asu.edu/theoretical-framework.html. [Accessed January 2017].

[22] D. Bolton, "Why I Choose PostgreSQL Over MySQL/MariaDB," March 2015. [Online]. Available: http://insights.dice.com/2015/03/19/why-i-choose-postgresql-

over-mysqlmariadb/. [Accessed July 2017].

APPENDIX A

CYPHER SCRIPT FOR CREATING GRAPH DATA & RELATIONSHIPS

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///imod.csv" AS row
MERGE (id:UniqueId{name:'Imod'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:Imod
{__id__:id.count,version:row.version,courseLocation:row.course_location,courseSemest
er:row.course_semester,creditHours:row.credit_hours,imodNumber:row.imod_number,na
me:row.name,numberOfSeats:row.number_of_seats,overview:row.overview,owner:row.o
wner_id,saved:row.saved,subjectArea:row.subject_area,url:row.url})


MATCH (n:Imod )
SET n.saved = (case n.saved when 't' then true else false end)
RETURN n

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///imod.csv" AS row
MATCH (imod:Imod {__id__: toInt(row.id)})
MATCH (user:ImodUser {__id__: toInt(row.owner_id)})
MERGE (user)-[:OWNS]->(imod);


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///imod_user.csv" AS row
MERGE (id:UniqueId{name:'ImodUser'})
ON CREATE SET id.count = 1 ON MATCH SET id.count = id.count + 1

CREATE(:ImodUser{__id__:id.count,version:row.version,email:row.email,firstName:ro
w.first_name,lastName:row.last_name,location:row.location,officeHours:row.office_hour
s,password:row.password,phoneNumber:row.phone_number,username:row.username,we
bPage:row.web_page})


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///kd.csv" AS row
MERGE (id:UniqueId{name:'KnowledgeDimension'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:KnowledgeDimension
{__id__:id.count,description:row.description,info:row.info})


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///dc.csv" AS row
```

```
MERGE (id:UniqueId{name:'DomainCategory'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:DomainCategory {__id__:id.count,name:row.name,priority:row.priority})

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///ld.csv" AS row
MERGE (id:UniqueId{name:'LearningDomain'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:LearningDomain {__id__:id.count,version:row.version,name:row.name})



USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///awcf.csv" AS row
MERGE (id:UniqueId{name:'ActionWordCategory'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:ActionWordCategory
{__id__:id.count,actionwordcategory:row.action_word_category,dc:toInt(row.domain_ca
tegory_id)})



USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///dc.csv" AS row
MATCH (dc:DomainCategory {__id__: toInt(row.id)})
MATCH (ld:LearningDomain {__id__: toInt(row.learning_domain_id)})
MERGE (dc)-[:LEARNINGDOMAIN]->(ld)
MERGE (ld)-[:LDHASDC]->(dc)

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///awc.csv" AS row
MATCH (awc:ActionWordCategory {__id__: toInt(row.id)})
MATCH (dc:DomainCategory {__id__: toInt(row.domain_category_id)})
MERGE (awc)-[:AWCHASDC]->(dc)
MERGE (dc)-[:DOMAINCATEGORY]->(awc)


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///learningobjective.csv" AS row
MERGE (id:UniqueId{name:'LearningObjective'})
ON CREATE SET id.count = 1
```

```
ON MATCH SET id.count = id.count + 1
CREATE (:LearningObjective {__id__:id.count,actionWord :row.action_word
,awc:toInt(row.action_word_category_id),condition:row.condition,criteriaAccuracy:row.
criteria_accuracy,criteriaQuality:row.criteria_quality,
criteriaQuantity:row.criteria_quantity,criteriaSpeed:row.criteria_speed,criteriaTypeId:toI
nt(row.criteria_type_id),custom_condition:row.custom_condition,definition:row.definitio
n,hideFromLearningObjectiveCondition:row.
hide_from_learning_objective_condition,imod:toInt(row.imod_id),indicator
:row.indicator,performance:row.performance})

MATCH (n:LearningObjective )
SET n.awc = (case n.awc when n.awc then toInt(n.awc) else toInt(n.awc) end)
RETURN n


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///learningobjective.csv" AS row
MATCH (lo:LearningObjective {awc: toInt(row.action_word_category_id)})
MATCH (awc:ActionWordCategory {__id__: toInt(row.action_word_category_id)})
MERGE (lo)-[:actionWordCategory]->(awc);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///learningobjective.csv" AS row
MATCH (lo:LearningObjective {imod: toInt(row.imod_id)})
MATCH (imod:Imod {__id__: toInt(row.imod_id)})
MERGE (lo)-[:imod]->(imod);

MATCH (p:ImodUser{__id__:1} )<-[:OWNER]-(n:Imod)-[:provides]-
>(lo:LearningObjective)-[rel:actionWordCategory]-
>(awc:ActionWordCategory{actionwordcategory:'Recognize'})-
[dec:DOMAINCATEGORY]->(dc:DomainCategory)-[led:LEARNINGDOMAIN]-
>(ld:LearningDomain{name:'Cognitive'}) RETURN n, awc, dec,led


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///lo_con_joinnew.csv" AS row
MATCH (lo:LearningObjective {__id__: toInt(row.learning_objective_id)})
MATCH (con:Content {__id__: toInt(row.content_id)})
MERGE (lo)-[:CONTENTS]->(con);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///contentg.csv" AS row
MERGE (id:UniqueId{name:'Content'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
```

```
CREATE (:Content
{__id__:id.count,imod:row.imod_id,priority:row.priority,preReq:row.preReq,parentCont
entId:row.parent_content_id,topicTitle:row.topic_title})

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///contentg.csv" AS row
MATCH (con:Content {imod: toInt(row.imod_id)})
MATCH (imod:Imod {__id__: toInt(row.imod_id)})
MERGE (imod)-[:HASCONTENT]->(con);


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///assessmentTechniquef.csv" AS row
MERGE (id:UniqueId{name:'AssessmentTechnique'})
ON CREATE SET id.count = 1
ON MATCH SET id.count = id.count + 1
CREATE (:AssessmentTechnique {__id__:id.count,assesmentype :row.assesmentype
,assessmentFeedbackId:toInt(row.assessment_feedback_id),assigncheck:row.assigncheck
,description:row.description,difficulty:row.difficulty,
duration:row.duration,favcheck:row.favcheck,is_admin:row.is_admin,procedure:row.pro
cedure,reference:row.reference,title:row. title,type:row.type,whenToCarryOut
:row.when_to_carry_out,whereToCarryOut:row.where_to_carry_out})



MATCH (n:AssessmentTechnique )
SET n.is_admin = (case n.saved when 'true' then true else false end)
RETURN n

MATCH (n:AssessmentTechnique )
SET n.assigncheck = (case n.saved when 'true' then true else false end)
RETURN n

MATCH (n:AssessmentTechnique )
SET n.favcheck = (case n.saved when 'true' then true else false end)
RETURN n

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///con_kd.csv" AS row
MATCH (con:Content {__id__: toInt(row.content_content_dimensions_id
)})
MATCH (kd:KnowledgeDimension {__id__: toInt(row.knowledge_dimension_id
)})
MERGE (con)-[:conHasKD]->(kd);
```

```
Match(kd:KnowledgeDimension)<-[:KNOWLEDGEDIMENSION]-
(at:AssessmentTechnique)<-[:assignedLearningObjective]-(lo:LearningObjective)-
[:CONTENTS]->(con:Content)-[:conHasKD]->(conKd:KnowledgeDimension) where
lo.imod=1 and kd.__id__ <> conKd.__id__ Return
lo.imod,lo.__id__,con.__id__,conKd.__id__,at.__id__,kd.__id__ ORDER BY lo.__id__

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///at_kd.csv" AS row
MATCH (at:AssessmentTechnique {__id__:
toInt(row.assessment_technique_knowledge_dimension_id
)})
MATCH (kd:KnowledgeDimension {__id__: toInt(row.knowledge_dimension_id
)})
MERGE (at)-[:KNOWLEDGEDIMENSION]->(kd);


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///at_ld.csv" AS row
MATCH (at:AssessmentTechnique {__id__:
toInt(row.assessment_technique_learning_domain_id
)})
MATCH (ld:LearningDomain {__id__: toInt(row.learning_domain_id
)})
MERGE (at)-[:ASSESSTECHHASLDOMAIN]->(ld);


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///at_dc.csv" AS row
MATCH (at:AssessmentTechnique {__id__:
toInt(row.assessment_technique_domain_category_id
)})
MATCH (dc:DomainCategory {__id__: toInt(row.domain_category_id
)})
MERGE (at)-[:ASSESSTECHHASDC]->(dc);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///at_lo.csv" AS row
MATCH (lo:LearningObjective {__id__: toInt(row.learning_objective_id
)})
MATCH (at:AssessmentTechnique {__id__: toInt(row.assessment_technique_id
)})
MERGE (lo)-[:assignedLearningObjective]->(at);

USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///awc.csv" AS row
```

```
MATCH (awc:ActionWordCategory {__id__: toInt(row.id)})
MATCH (dc:DomainCategory {__id__: toInt(row.domain_category_id)})
MERGE (awc)-[:AWCHASDC]->(dc)
MERGE (dc)-[:DOMAINCATEGORY]->(awc)


USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///dc.csv" AS row
MATCH (dc:DomainCategory {__id__: toInt(row.id)})
MATCH (ld:LearningDomain {__id__: toInt(row.learning_domain_id)})
MERGE (dc)-[:LEARNINGDOMAIN]->(ld)
MERGE (ld)-[:LDHASDC]->(dc)
```